195 PTAS.
(IVA Incluido)
111
mi computer
CURSO PRACTICO DEL ORDENADOR PERSONAL,
EL MICRO Y EL MINIORDENADOR
P.V.P. Canarias, Ceuta y Melilla 185 Ptas.
BLACK JACK PAYS
Dealer must stand on 17 and must draw
INSURANCE PAYS 2 TO 1
9 788485 822836
101111
Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen X-Fascículo 111

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 209 80 22 - Télex: 93392 EPPA

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 120 fascículos de aparición semanal, encuadernables en diez volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-183-6 (tomo 10)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 048603
Impreso en España-Printed in Spain-Marzo 1986

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (120 fascículos más las tapas, guardas y transferibles para la confección de los 10 volúmenes) son las siguientes:

a) Un pago único anticipado de 27 105 ptas. o bien 10 pagos trimestrales anticipados y consecutivos de 2 711 ptas. (sin gastos de envío).

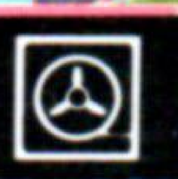
b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

No se efectúan envíos contra reembolso.





Modelo personal

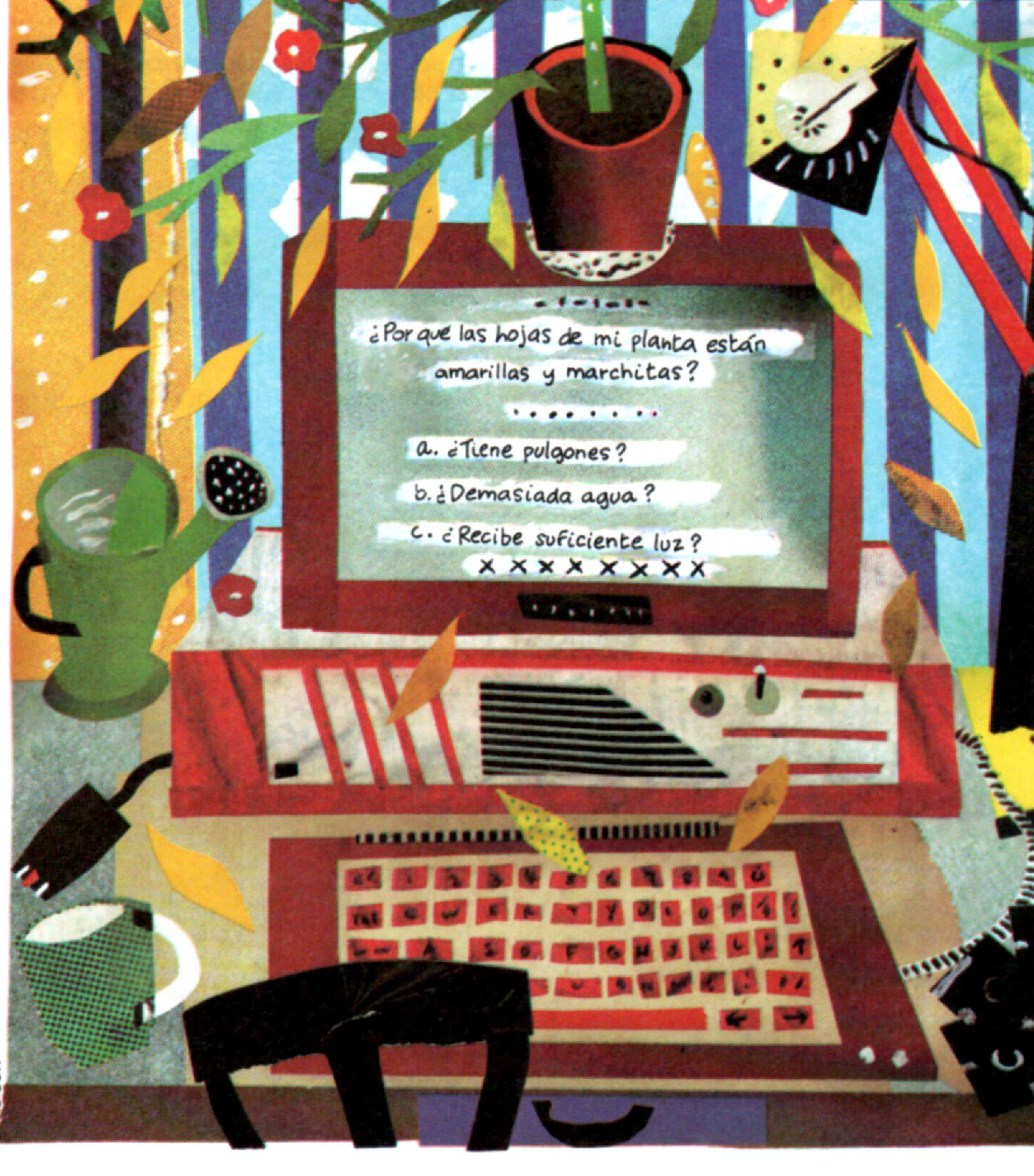
"Expert-Ease" es un programa que permite al usuario novel crear un sistema experto a su medida

Por definición los sistemas expertos tienden a aplicarse en campos determinados, tales como diagnóstico médico o reconocimiento geológico. Pero el generador de sistemas expertos ofrece al usuario la posibilidad de aplicar esta tecnología *soft* a problemas más inmediatos que, de lo contrario, no garantizarían el costo de preparar un sistema a medida. Un buen ejemplo de este enfoque es el que proporciona *Expert-Ease*, de Thorn-EMI Software, que permite crear un sistema experto propio (denominado *modelo*) y ejecutarlo en un IBM PC o un ACT Sirius, con un mínimo de 128 K de RAM.

Tal como mostramos en la página 1781, la parte de un sistema experto que realiza todo el trabajo es el motor de inferencias: el módulo de proceso lógico. Es la programación de esta sección lo que normalmente pone el enfoque de un sistema experto fuera del alcance de la mayoría de los usuarios, a menos que el gasto de programar la aplicación pretendida se considere absolutamente imperativo. Pero en realidad *Expert-Ease* crea un motor de inferencias para el usuario, de modo que todo el trabajo pesado lo realiza el ordenador en lugar de la persona que está creando el modelo.

Existen cuatro enfoques principales para diseñar el motor de inferencias de un sistema experto: el formalismo basado en reglas, empleando una estructura IF...THEN; las redes semánticas, basadas en conjuntos y subconjuntos ("batería descargada", por ejemplo, es un miembro del conjunto "fallo eléctrico"); ventanas o marcos, que se preparan como una base de datos tradicional con la excepción de que los valores especificados de los campos especificados hacen que el sistema saque conclusiones sobre el registro; y cláusulas cuerno, la lógica de predicado en la que se basa el PROLOG. *Expert-Ease* utiliza un quinto método automatizado: el *sistema analógico de aprendizaje de conceptos* (*Analogue Concept Learning System*: ACLS).

El ACLS se desarrolló como módulo independiente en el laboratorio de inteligencia artificial de la Universidad de Edimburgo por el personal y los consultores de Intelligent Terminals Ltd, una empresa propiedad del profesor Donald Mitchie, a quien se considera una de las primeras autoridades del mundo en materia de inteligencia artificial. El estudio del funcionamiento del ACLS está fuera del alcance de este capítulo, pero para quienes estén interesados en el mismo, se desarrolló a partir del *Iterative Dichotomiser 3* (dicotomizador iterativo: ID3), un programa de dominio público escrito en PASCAL.



Chris Corr

Guía de experto
Expert-Ease, un generador de sistemas expertos producido por Thorn-EMI, permite elaborar sistemas expertos a la medida de forma rápida y económica para distintas aplicaciones. Particularmente útil en diagnóstico y análisis de problemas, la potencia del sistema experto (aunque previamente restringido a áreas con un elevado perfil financiero) ya se puede aplicar a situaciones más domésticas

Un problema sencillo

Los modelos de *Expert-Ease* en realidad se crean hacia atrás, en el sentido de que usted primero debe decidir cuáles son todas las posibles conclusiones y luego decidir cuáles son las preguntas que es necesario que responda el usuario, y por último decidir qué respuestas conducirán a qué decisiones. A modo de ejemplo de la estructura del *Expert-Ease*, veamos una aplicación muy simple: resolver el problema de si debemos ir o no al partido de fútbol de esta tarde. A pesar de sus evidentes limitaciones, al abordar este problema demostramos el potencial del sistema para aplicaciones en áreas más exigentes de diagnóstico y toma de decisiones.

Las dos conclusiones posibles para este problema son, obviamente, "ir" y "no ir", de modo que lo siguiente que hemos de hacer es decidir qué preguntas necesitamos formular o, por decirlo de otro modo, qué factores son importantes para tomar la decisión. Nuevamente, para simplificar, vamos a suponer que sólo estamos interesados en el buen tiempo y en tener suficiente tiempo. Comencemos por elaborar una tabla de decisión:

Ejemplo/ Pregunta	¿Buen tiempo?	¿Tiempo suficiente?	Decisión
1.	S	S	IR
2.	S	N	NO
3.	N	*	NO
4.	N	*	NO

Es claro que si las respuestas son negativas, la decisión es "no". Podemos alterar la tabla así:

Ejemplo/ Pregunta	¿Buen tiempo?	¿Tiempo suficiente?	Decisión
1.	S	S	IR
2.	S	N	NO
3.	N	S	NO
4.	N	N	NO

donde el asterisco significa "no preocuparse". De manera que si la respuesta a la primera pregunta es negativa, es innecesario formular la segunda.

Así es, en realidad, cómo requiere el *Expert-Ease* que entremos la información. En primer lugar, crea una tabla marco que contiene las preguntas y los posibles resultados (conocida como *pantalla de atributos*) y después rellena esta pantalla con respuestas ejemplo (conocida como *pantalla de ejemplos*). La única diferencia es que el *Expert-Ease* no nos limita a respuestas de sí o no: podemos tener tantas respuestas como queramos. Las preguntas se diseñan en un formato de múltiple opción, y rellenamos los números en la pantalla de ejemplos. Una vez creada la pantalla de ejemplos, el *Expert-Ease* utiliza el ACLS para inducir la lógica de la tabla de decisión para crear una "regla". Ésta se aplica cuando se ejecuta el módulo.

El ACLS (que en el paquete se denomina *rutina de inducción*) comprueba nuestra tabla en busca de conflictos y nos los señala (tales como: formadas 11 reglas nudo, ejemplos 1 5 6 contradicen la regla).

Expert-Ease es indudablemente un programa sumamente potente y sofisticado. Para la mayoría de las aplicaciones comunes, los modelos que crea responden a cualquier sistema experto especializado. Además, se pueden encadenar modelos entre sí, de

Creación de un modelo

La primera etapa en la creación de un modelo *Expert-Ease* es diseñar el marco en el cual se basará el esquema de decisión. *Expert-Ease* lo denomina *pantalla de atributos* (que se llama pulsando A de la pantalla archivadora). Para crear esta pantalla, hemos de decidir qué preguntas necesitamos responder y en qué orden. Damos nombres a cada una de estas preguntas y luego se visualizan como títulos de columnas. A continuación hemos de elaborar cuáles son todas las decisiones posibles (o sugerencias). A éstas se les dan nombres y se visualizan como títulos de filas en el lado derecho de la pantalla. En el caso de nuestro modelo de transporte, las preguntas que hemos decidido formular conciernen al dinero, los pasajeros, las mercancías, la velocidad y el esfuerzo (si queremos o no transporte motorizado). Las sugerencias posibles son transporte público, bicicleta, Sinclair C5, motocicleta y automóvil.

Una vez que hemos creado estos títulos, podemos entrar el texto para las preguntas y sugerencias pulsando T (para visualizar el texto de acompañamiento), seguida de ' (una sola comilla) para crear o editar (el editor está al nivel del editor en pantalla Commodore). Una vez que hemos entrado el texto, CTRL-C lo almacena y Escape desestima el texto editado. Parecería no haber límites (aparte de la memoria) respecto a la cantidad de texto que se puede entrar; cuando se ejecute el modelo, lo que exceda de una pantalla completa se dividirá automáticamente en secciones del tamaño de la pantalla. Sin embargo, manteniendo las preguntas en una sola pantalla, el sistema resulta más rápido y más fácil de usar.

La pantalla de ejemplos es el verdadero esquema de decisión. Se lo llama así porque el usuario muestra al *Expert-Ease*, mediante ejemplos, cómo se han de tomar las decisiones. Por consiguiente, el ejemplo 1 es un usuario que responde 2 (no) a la pregunta del dinero, indicando que carece de capital para gastar. Puesto que las respuestas a las otras preguntas son irrelevantes, colocamos un asterisco (el símbolo de "no preocuparse") bajo cada una de las otras columnas. El *Expert-Ease* se saltará entonces todas las preguntas que lleven debajo un asterisco.

En el ejemplo 2 necesitamos formular cada pregunta, porque sólo cuando tenemos la respuesta a la última pregunta sabemos si el usuario requiere una bicicleta o un C5. El ejemplo 3 muestra las respuestas que conducen a recomendar un C5.

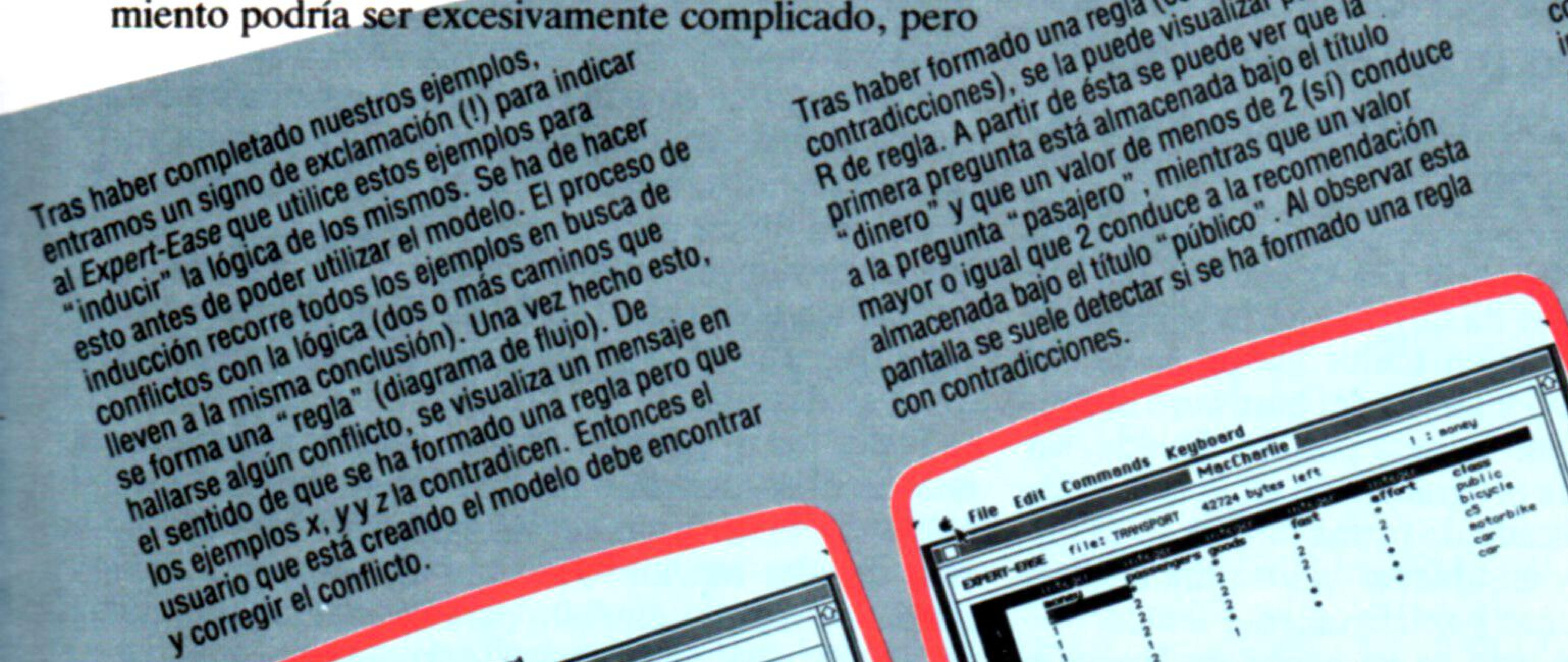
Aplicaciones fáciles

El *Expert-Ease* es ideal para crear modelos simples, en especial aquellos que quizá se hayan de modificar para adecuarse al futuro desarrollo.

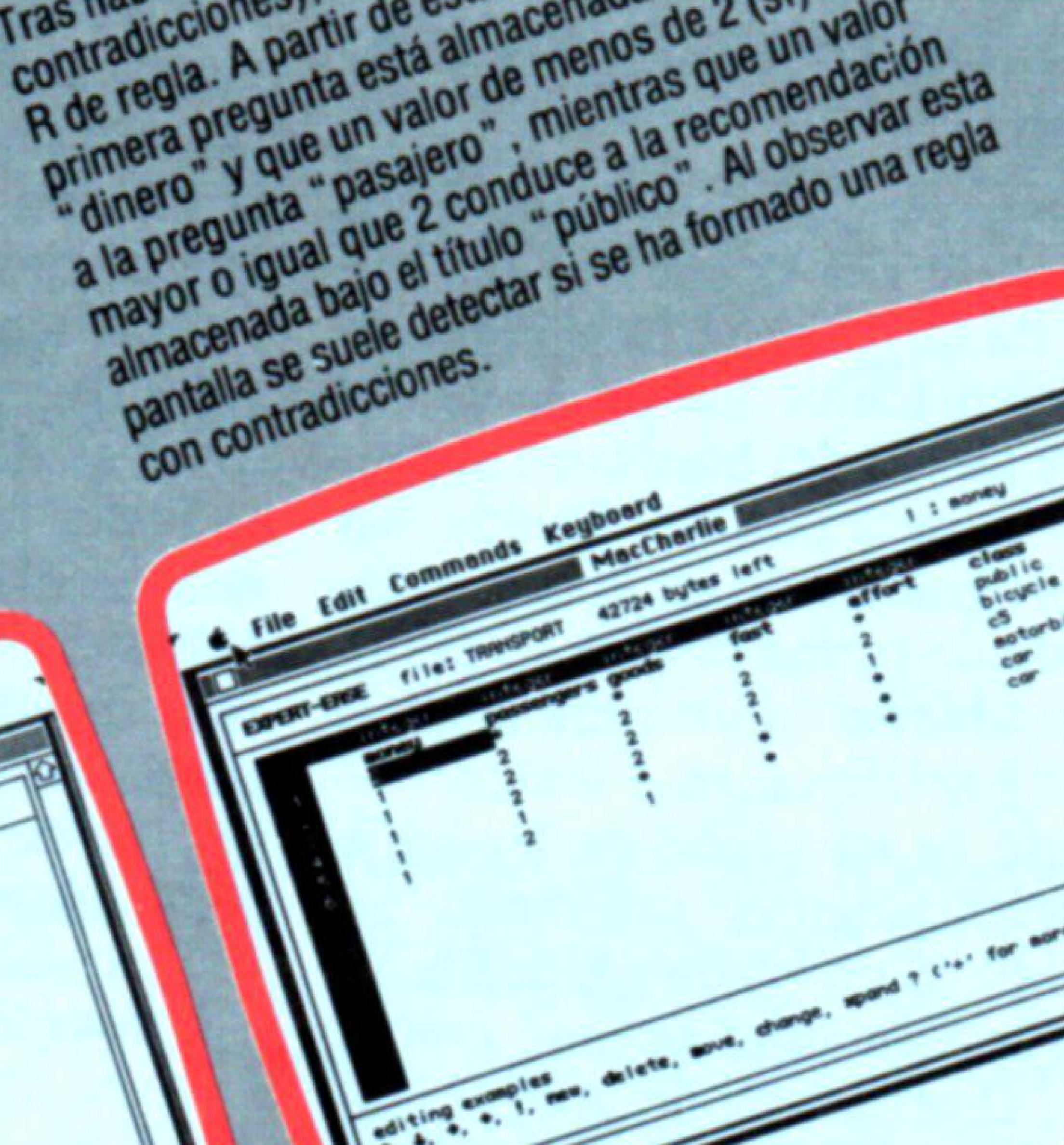
Aplicaciones de crédito

Los sistemas expertos pueden ser de enorme utilidad en el área de proceso de solicitudes de crédito. En uno de los extremos del espectro está la pequeña empresa, que necesita decidir si conceder o no a un cliente una facturación a 30 días, pago contra entrega o pago contra pedido. En el otro extremo está el banco que estima solicitudes de préstamos de millones de pesetas. El principio es el mismo en ambos casos. La concesión de créditos supone necesariamente riesgo; la única cuestión es si el riesgo entra o no dentro de límites aceptables. La tarea del controlador de créditos o gestor de préstamos es inicialmente evaluar el riesgo y, en segundo lugar, decidir si éste entra dentro de los límites aceptables para la empresa o el banco. Crear un modelo *Expert-Ease* para llevar a cabo la misma función es simplemente cuestión de categorizar los criterios utilizados para evaluar la "conveniencia" del crédito, idear preguntas para cada uno y utilizar las respuestas para evaluar el

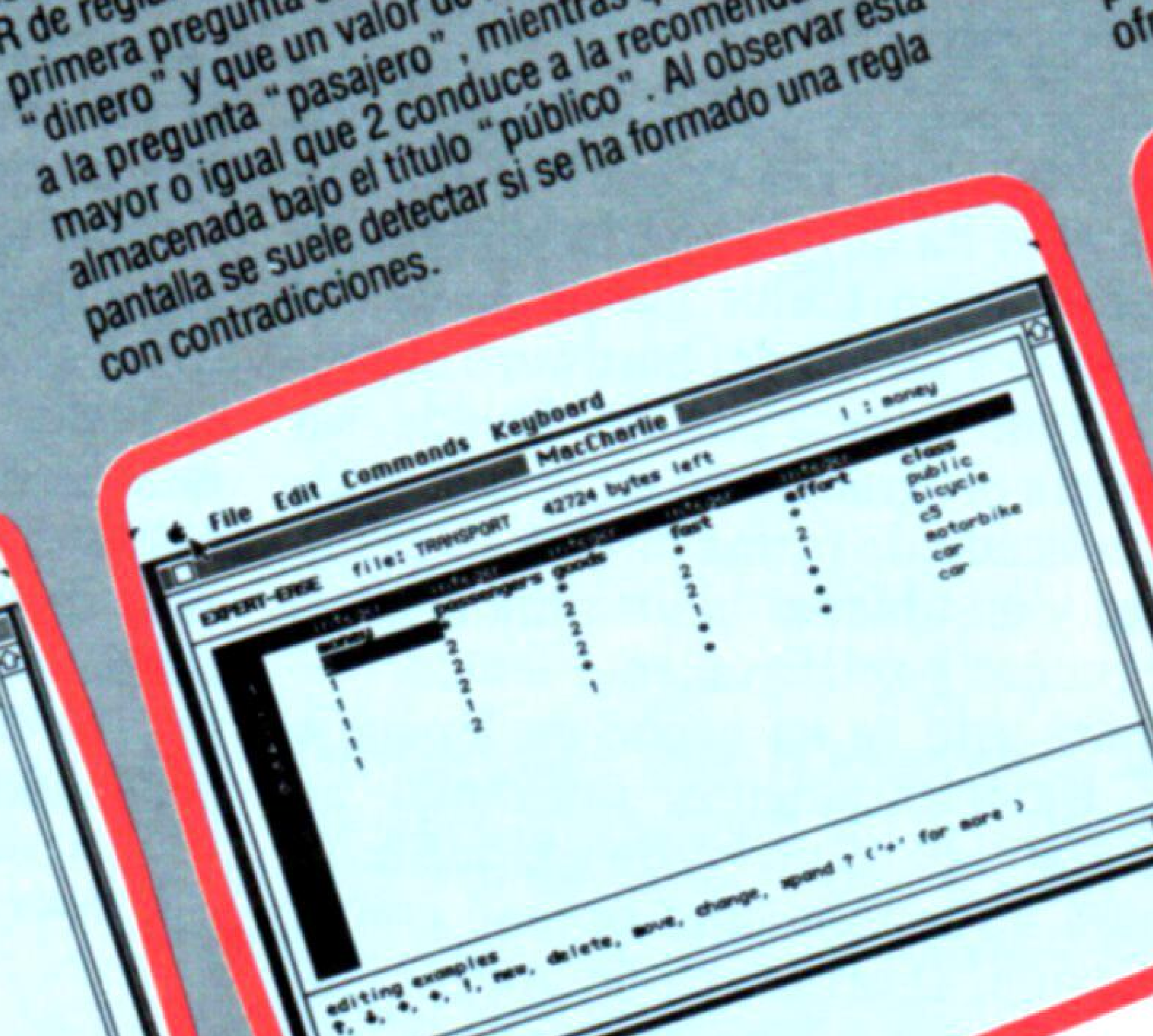
modo que es factible descomponer problemas complejos en piezas manejables. La principal diferencia entre un modelo *Expert-Ease* y un sistema experto escrito a medida es que el primero no explica su propio razonamiento. Usted, por supuesto, puede crear sus propias explicaciones en el texto de la conclusión, pero al usuario no se le presenta el verdadero camino de decisión del modelo. Esto no constituye desventaja en sistemas diseñados para que los utilicen no expertos, puesto que el razonamiento podría ser excesivamente complicado, pero sí es significativo en modelos diseñados para ayudar a los expertos, quienes quizá deseen cuestionar una parte o la totalidad del razonamiento.



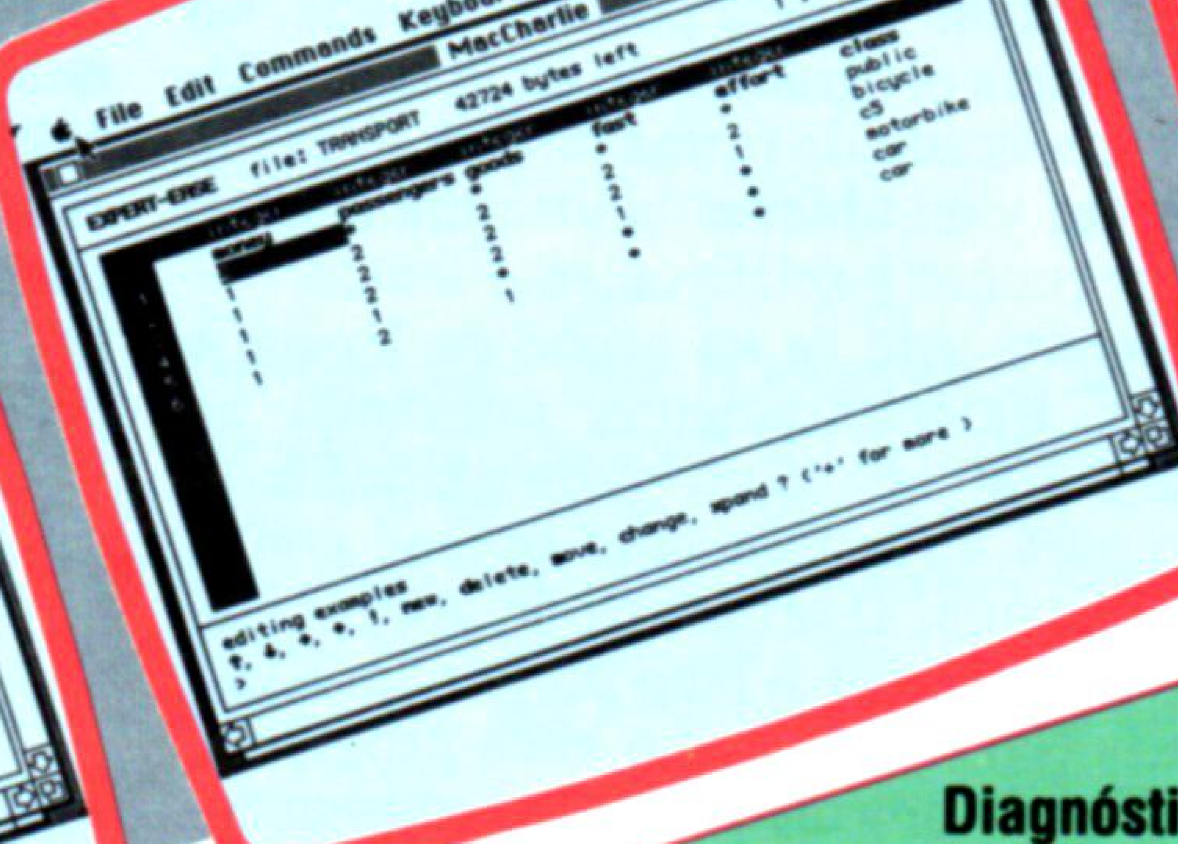
Tras haber completado nuestros ejemplos, entramos un signo de exclamación (!) para indicar al *Expert-Ease* que utilice estos ejemplos para "inducir" la lógica de los mismos. Se ha de hacer esto antes de poder utilizar el modelo. El proceso de inducción recorre todos los ejemplos en busca de conflictos con la lógica (dos o más caminos que lleven a la misma conclusión). Una vez hecho esto, se forma una "regla" (diagrama de flujo). De hallarse algún conflicto, se visualiza un mensaje en el sentido de que se ha formado una regla pero que los ejemplos x, y y z la contradicen. Entonces el usuario que está creando el modelo debe encontrar y corregir el conflicto.



Tras haber formado una regla (con o sin contradicciones), se la puede visualizar pulsando la R de regla. A partir de ésta se puede ver que la primera pregunta está almacenada bajo el título "dinero" y que un valor de menos de 2 (sí) conduce a la pregunta "pasajero", mientras que un valor mayor o igual que 2 conduce a la recomendación almacenada bajo el título "público". Al observar esta pantalla se suele detectar si se ha formado una regla con contradicciones.



Usted puede disponer de breves descripciones de las instrucciones del *Expert-Ease* en cualquier momento dado pulsando H o ?. La excepción a esto se produce cuando usted se halla en mitad de la tarea de rellenar un ejemplo en la pantalla de muestra; en este caso primero es necesario completar el ejemplo. Se le preguntará con qué instrucción le agradaría una ayuda. Pulsando H obtiene un resumen de una línea de todas las instrucciones válidas en la pantalla. Pulsando la primera letra de cualquier otra instrucción se le ofrecerá una descripción algo más larga



Visualizaciones de Macintosh con MacCharlie

riesgo. Un modelo simplificado de préstamo bancario podría ser más o menos así:

Seguridad	Deuda irrecup.	Girado en descub.	Ingresos regulares	Resultados
S	*	*	*	OK
N	S	*	*	NO
N	N	S	S	SOMETER
N	N	S	N	NO
N	N	N	N	SOMETER
N	N	N	S	OK

Una ejecución típica podría ser como ésta:

¿Ofrece seguridad el cliente? No
¿Tiene el cliente algún antecedente de deuda irrecuperable? No
¿El cliente ha girado alguna vez en descubierto sin autorización? Sí
¿Tiene éste ingresos regulares comprobables? Sí
Por favor someter esta solicitud a un gestor de préstamos.

De este modo, las solicitudes directas las puede manejar un empleado subalterno utilizando el sistema experto, mientras que las solicitudes que requieran una consideración más detallada se someterán a un gestor de préstamos o gerente. Si el banco modificara sus criterios generales para la concesión de préstamos, el modelo se podría modificar rápidamente.

Diagnósticos

Los sistemas expertos representan una solución ideal para cualquier clase de problema de diagnóstico. Los problemas complejos, tales como los que surgen en medicina, obviamente exigen grandes cantidades de datos junto con complejas reglas que impliquen sutiles distinciones, y, por lo tanto, necesitan de un sistema escrito especialmente. Sin embargo, *Expert-Ease* puede tratar fácilmente tareas menos complicadas.

Por ejemplo, una empresa de helicópteros que lleva y trae a su trabajo a los empleados de una instalación petrolífera utiliza el *Expert-Ease* como ayuda en el servicio de mantenimiento rutinario de su flota de aparatos. Normalmente, cuando se trae un helicóptero para un servicio de rutina menor, se requiere un ingeniero superior para inspeccionarlo y determinar el tipo de servicio requerido. Una vez hecho esto, los mecánicos pueden llevar a cabo el trabajo. Sin embargo, utilizando el *Expert-Ease* un mecánico puede determinar el trabajo que es necesario llevar a cabo sin consultar al ingeniero superior. Éste se limita a realizar su inspección normal una vez que se ha efectuado el trabajo.

El mismo principio se puede ampliar fácilmente al mantenimiento y reparación de todo tipo de equipos mecánicos, eléctricos y electrónicos, incluyendo, por supuesto, ordenadores. De hecho, el *Expert-Ease* se puede utilizar para detectar fallos de software o hardware. Por ejemplo, cuando un cliente solicita servicio técnico, los ingenieros han de pasar un valioso tiempo detectando el fallo de la máquina. No obstante, con el *Expert-Ease* esta tarea se podría dejar en manos de personal menos cualificado, planteando a los ingenieros sólo los problemas graves

Lenguaje puente

Iniciamos una serie dedicada al c, un lenguaje que llena el vacío existente entre los lenguajes de alto y bajo nivel

Recientemente, gran parte del desarrollo de lenguajes de alto nivel se ha dirigido hacia la creación de lenguajes que reflejen mejor los procesos del pensamiento humano y alejen del hardware al programador en la mayor medida posible. Queda, sin embargo, muchísima programación en la cual el informático está implicado de forma directa en manipular el hardware y en obtener la máxima eficacia de máquinas pequeñas y relativamente lentas.



Marcus Wilson-Smith

C de calidad
El Hisoft-c se ejecuta en los ordenadores Amstrad y Spectrum. La implementación es moderadamente estándar, con la grave excepción de que no se incluyen números de punto flotante. No obstante, Hisoft tiene entre sus planes producir en un futuro cercano una versión con punto flotante. Otro punto objetable lo constituye el manual, que, si bien es exhaustivo, está muy mal dispuesto y no es adecuado para principiantes. Siempre y cuando el usuario tenga ya alguna experiencia previa o una documentación adicional, éste es un paquete excelente

Tradicionalmente esto se ha hecho en lenguaje ensamblador o, esporádicamente, utilizando un lenguaje como el FORTRAN en un hardware que haya sido diseñado para un rendimiento óptimo con el lenguaje. No obstante, el problema de utilizar lenguaje ensamblador es que si bien proporciona una máxima eficacia (en manos de un buen programador), es específico de una máquina. Por consiguiente, durante mucho tiempo ha existido la necesidad de un lenguaje sencillo a un nivel relativamente bajo que fuera fácilmente implementable en una gran variedad de máquinas, combinando las estructuras y la conveniencia de un lenguaje de alto nivel con un control directo y eficaz sobre el hardware.

El primer lenguaje del cual se pudo afirmar que reunía estos requisitos fue el BCPL, desarrollado por Martin Richards en la Universidad de Cambridge. Ken Thompson y otros colaboradores de Bell Labs aplicaron muchas de las ideas del BCPL para un lenguaje denominado B. En aquella época estaban desarrollando el sistema operativo Unix y para ello necesitaron un lenguaje de alto nivel. Kernighan y Ritchie convirtieron al B en el c y en 1978 publicaron el libro que lo define, *The C Programming Language*, que sirve como estándar para las implementaciones de c (hasta que se llegue a un acuerdo para un nuevo estándar internacional).

También se estaba desarrollando el Unix y desde entonces el lenguaje y el sistema operativo han estado íntimamente relacionados. Las 13 000 líneas de código que componen el *kernel* o núcleo del Unix contienen apenas 800 líneas de ensamblador, siendo de c el resto de las líneas. Otro punto a favor del c es que se trata de un lenguaje relativamente pequeño, lo que no sólo lo vuelve fácil de aprender y utilizar, sino que también conlleva que los compiladores sean pequeños y fáciles de escribir, de modo que el lenguaje se puede implementar en una amplia variedad de máquinas. Además, dado que el lenguaje ya es de bajo nivel, el código producido es compacto y rápido. El resultado de esto es que no sólo el Unix y las aplicaciones Unix se escriben en c, sino también el grueso de los sistemas operativos MS-DOS y CP/M.

Existen compiladores de c para casi todos los micros, así como para ordenadores centrales, y su uso ha permitido que las casas de software produzcan programas que se puedan trasladar con facilidad entre máquinas tan disímiles como el Apple Macintosh y el IBM PC. A lo largo de esta serie nos ceñiremos a la versión estricta del lenguaje de Kernighan y Ritchie, y allí donde sea conveniente, como cuando consideremos bibliotecas de funciones, daremos por sentado el empleo del Unix, si bien la mayoría de las otras versiones de c seguirán esta versión lo más estrechamente posible.

Un programa en c se construye mediante funciones de definición, cada una de las cuales toma varios argumentos, valores o punteros de valores, que la función utiliza para producir un valor o puntero que se devuelve como resultado. Cada función se invoca simplemente escribiendo su nombre. El valor devuelto se puede utilizar, aunque en algunos casos se descarta.

Todos los programas en c empiezan ejecutando una función llamada main, de modo que el nivel más exterior de un programa debe ser una definición de esta función. He aquí un programa muy simple pero muy completo para efectuar la primera prueba de cualquier nuevo lenguaje o sistema de ordenador:


```
main ()
{
   printf("hola mundo\n");
}
```

Incluso este sencillo programa nos pone de relieve algunas importantes características del lenguaje. La definición de una función consiste en el nombre de la función seguido por una lista de sus argumentos encerrados entre paréntesis. En este caso, sucede que main no tiene argumentos pero, aun así, se han de colocar los paréntesis.

El cuerpo de la definición de función consiste en una secuencia de sentencias (sólo una, en este caso) encerrada entre corchetes ({}) y cada sentencia termina con un punto y coma. Los corchetes actúan de modo similar al Begin...End del PASCAL, en tanto y en cuanto se los puede utilizar para encerrar cualquier secuencia de sentencias y declaraciones para conformar una sentencia compuesta, que a su vez se puede utilizar en cualquier posición en la que se pueda usar una sentencia simple. El punto y coma se utiliza de manera diferente al PASCAL, pero la regla es mucho más simple: cada sentencia completa debe ir seguida de un punto y coma. La única sentencia de este programa es una llamada a una función de biblioteca, printf.

La entrada/salida no está definida estrictamente en c, porque varía mucho de una máquina a otra. En cambio, queda en manos de cada una de las implementaciones el proporcionar las E/S que se requieran en una biblioteca, si bien la mayoría son fieles al estándar. El valor devuelto por la función printf no sirve, de modo que se lo ignora. Puede haber un número cualquiera de argumentos para printf, ya sea numéricos, variables de tipo carácter o series (como en este caso). Asimismo, printf no genera automáticamente un retorno, pero el c proporciona una cantidad de símbolos especiales para caracteres de control, todos los cuales comienzan con una barra invertida (\). El símbolo de retorno (o línea) es \n (véase la tabla *Secuencias Escape*).



El siguiente programa daría exactamente el mismo resultado que el primero:

```
main ()
{
   printf('hola ');
   printf('mundo');
   printf('\n');
}
```

El c es un lenguaje con variables tipificadas, es decir, cada variable se debe declarar como de un tipo determinado. Siempre están disponibles los siguientes tipos:

char	–caracteres, ocupando un único byte.
short	–enteros cortos.
int	–enteros normales.
long	–enteros largos.
float	–números (reales) con punto flotante.
double	–números con punto flotante de doble precisión.

Las declaraciones de variables se efectúan dando el nombre del tipo seguido por una lista de nombres de las variables que usted quiere que sean de ese tipo. Se deben declarar todas las variables, si bien, como ya veremos, las declaraciones pueden producirse casi en cualquier punto del programa. Además, los nombres de las variables pueden ser de cualquier longitud, pero por lo general sólo son significativos los ocho primeros caracteres.

Asignación y aritmética

Como es habitual, no se admiten espacios en nombres de variables, pero se puede utilizar el carácter de subrayado. Los nombres no pueden empezar con un dígito ni tampoco con un subrayado, pero esto sólo obedece a un posible conflicto con funciones de biblioteca. El tipo de letra (mayúscula o minúscula) es significativo, de modo que, por ejemplo, A es una variable distinta de a. No está especificada la cantidad total de bytes para cada tipo numérico, de modo que pueden variar para cada implementación. Típicamente, sin embargo, un entero short será de ocho bits, int será de 16 bits y long será de 32 bits.

La asignación y la aritmética siguen convenciones bastante normales, utilizándose el signo de igualdad (=) como operador de asignación. Los operadores aritméticos usuales +, –, * y / se utilizan junto con %, que da el módulo. De modo que:

```
int x,y,z
...
z=x%y
```

Las expresiones y las asignaciones pueden incluir cualquier variedad de tipos numéricos, y char, que a estos fines se considera como un entero de un byte que contiene el código ASCII del carácter. El c lleva a cabo de forma automática todas las conversiones de tipos necesarias convirtiendo los tipos *inferiores* a *superiores*. Es interesante destacar aquí que todas las operaciones de punto flotante normalmente se efectúan en doble precisión, aun cuando todos los operandos sean meramente float. También es posible forzar un cambio de tipo mediante un "matiz" en el cual el nombre de variable va precedido por el nuevo tipo entre paréntesis. Por ejemplo:

```
int n;
float f;
. . . .
f=sqrt((double)n);
```

toma una copia de doble precisión del valor de n, dado que la función raíz cuadrada espera que su argumento sea de tipo double, pero el verdadero valor de n queda inalterado.

El c proporciona dos operadores adicionales muy útiles, que son los operadores de incremento y decremento ++ y ––. De modo que, por ejemplo, ++a significa incrementar y ––a significa decrementar el valor de a *antes* de que se ejecute la sentencia actual; a++ significa incrementar y a–– significa decrementar el valor de a *después* de que se ejecute la sentencia actual. Por consiguiente:

```
int a,b
. . .
b=1;
a=b++;/* deja 1 en a y 2 en b*/
```

Mientras que:

```
b=1
a=++b:/* deja 2 tanto en a como en b*/
```

Otro detalle es que la asignación, al igual que todas las otras operaciones del c, se trata como una función y devuelve un valor, a saber, el valor que se está asignando. Éste es un valor perfectamente legítimo para utilizar en cualquier otra expresión; de modo que, por ejemplo, podemos escribir:

```
x=y=z;
```

que, efectivamente, asigna el valor de z tanto a x como a y.

Ya hemos mencionado que la entrada/salida es manejada por funciones de biblioteca y que puede variar de una implementación a otra. No obstante, evidentemente es difícil escribir muchos programas sin algún tipo de E/S, de modo que terminaremos este primer capítulo considerando de modo sucinto las dos funciones principales para E/S de terminal. Una de ellas ya la hemos visto: printf, que formatea y produce valores de diversos tipos en el dispositivo de salida estándar (normalmente la pantalla). Su sintaxis completa es:

```
printf(serie_de_control, arg1,arg2,.....);
```

donde los argumentos pueden ser series o variables de cualquiera de los tipos de datos básicos.

La serie_de_control puede contener secuencias comunes de caracteres, que simplemente se producen tal como son, pero también contiene un especificador de formato para cada uno de los argumentos que, dicho sea de paso, también proporciona la cantidad de argumentos. Un especificador de formato empieza con un signo % y termina con uno de los caracteres de conversión, como vemos en la tabla *Conversión de formatos*. Entre los argumentos puede aparecer –, para especificar el ajuste a la izquierda en el campo de salida, y un número para especificar la anchura del campo en el cual se ha de imprimir el valor. Opcionalmente, éste puede ir seguido por un punto y otro número que especifique la cantidad de caracteres a imprimir realmente (en el caso de una serie) o la cantidad de posiciones tras el punto decimal (en el caso de un número float o double precision).

Secuencias Escape

\n	Nueva línea
\t	Tab horizontal
\b	Retroceso
\r	Retorno de carro
\f	Nueva página
\'	Produce el car. apóstrofo (de lo contrario, utilizado para encerrar un car.)
\"	Produce el car. del signo de comillas (de lo contrario, usado para encerrar series)
\d	Donde d es un dígito octal. Produce el car. con el cód. ASCII correspondiente
\xh	Donde h es un dígito hexa. Produce el car. con el cód. ASCII correspondiente
\e	Escape
\v	Tab vertical

Conversión de formatos

Carácter	Tipo de argumento	Acción
d	int	formato decimal
o	int	formato octal
x	int	formato hexa
u	int	formato decimal sin signo
c	char/int	salida como car. simple.
s	*char	salida como serie
e	float/ double	formato mantisa-exponente
f	float/ double	formato decimal normal
g	float/ double	cualquiera de d, e o f que sea más corto
%		produce el carácter %

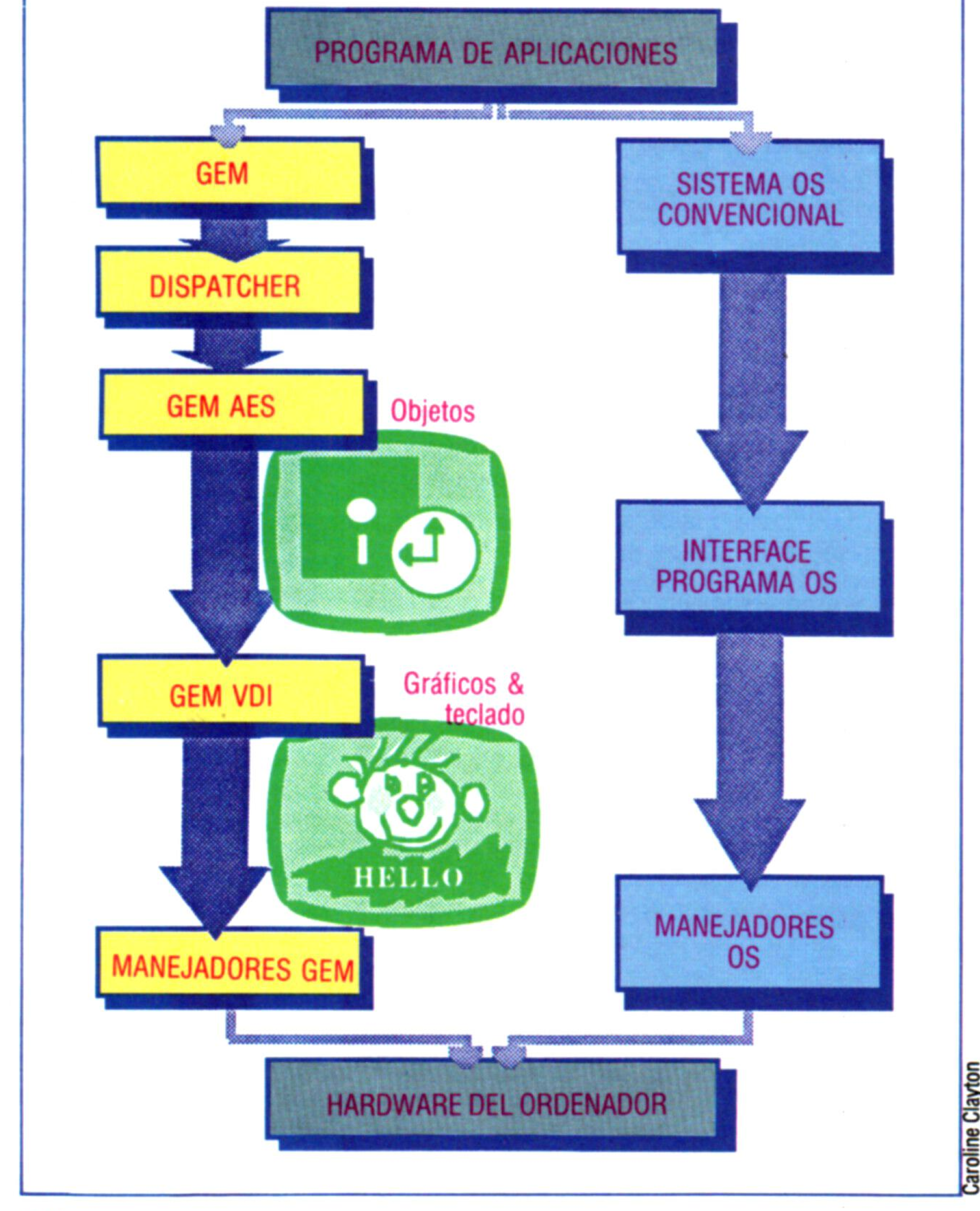
Trabajar con ventanas

Finalmente estudiaremos algunas de las facilidades operativas y las estructuras de programa del GEM

Rutas diferentes
La imagen "final" de fácil uso del GEM contradice su innata complejidad. La VDI (interface de dispositivo virtual) permite escribir código independiente de la máquina y se divide en dos partes. Una, el GDOS, cumple un papel de OS casi tradicional procesando las entradas del usuario, y la otra (los manejadores de dispositivo) se adapta a la determinada configuración de hardware que se esté utilizando. El módulo AES (*applications environment services*: servicios de entorno de aplicaciones) contiene diversas subrutinas de alto nivel para tratamiento de pantalla y periféricos

Si bien es probable que Windows, de Microsoft, llegue a ser un producto importante para sistemas de gestión, existen muchas más probabilidades de que el usuario personal encuentre el GEM más asequible. El GEM ya viene empaquetado con el nuevo Atari 520ST basado en el Motorola 68000 y también está apareciendo en máquinas tales como el Apricot y el RML Nimbus.

Así como los programadores de aplicaciones convencionales pueden escribir para un sistema operativo en vez de para una determinada máquina, los WIMP basados en GKS proporcionan una forma estandarizada de direccionar los dispositivos gráficos subyacentes. Un programa escrito para GEM, que se basa en GKS, es intencionadamente portable para todo hardware que posea el kernel necesario para software de gráficos. Los sistemas de kernel para gráficos tales como el GEM, GSX y GKS actúan a modo de un caparazón de software alrededor tanto del sistema existente como del hardware para gráficos. La entrada desde el teclado y el tratamiento de archivos se controlan mediante el sistema operativo anfitrión, como antes, pero toda manipulación de gráficos es interceptada por el "sistema operativo" de gráficos.



Caroline Clayton

Digital Research alude a su producto como GEM DOS, pero el GEM propiamente dicho en realidad no lleva a cabo ninguna de las funciones DOS normales. Las instrucciones provenientes de dispositivos que no sean el teclado (un ratón, p. ej.) se traducen a llamadas equivalentes a las rutinas del sistema existentes. La actualización de la visualización en pantalla u otros dispositivos gráficos la maneja el sistema GSX subyacente: el corazón del GEM. Este proceso es adicional al intérprete de líneas de instrucciones del sistema convencional.

Siempre que una operación del sistema requiera una visualización de datos, la habitual visualización de carácter a carácter se sustituye por un flujo de datos gráficos por mapa de bits hacia el manejador del dispositivo apropiado. Cada dispositivo de hardware posee su propio manejador con unas características adecuadas a la función de ese dispositivo (entrada, salida, resolución, etc.). Se pueden instalar manejadores de dispositivos en la medida que sea necesario. Este plano de comunicación software entre el sistema y dispositivos de hardware específicos es lo que se denomina *interface de dispositivo virtual (virtual device interface*: VDI).

Programar para el GEM propiamente dicho es un proceso complejo. En ausencia de una poderosa biblioteca de procedimientos para tratar eventos, activar y desactivar ventanas (restableciendo los contenidos anteriores), etc., el programador de GEM debe construir cualquier rutina requerida llamando a las primitivas VDI GEM/GSX. Por ejemplo, la trivial tarea de crear una ventana y escribir algún texto en ella, como en PRINT 'Hola!' en BASIC, en GEM entraña un considerable esfuerzo de programación. En primer lugar, se deben inicializar la posición y dimensiones de la ventana a unos valores por defecto. Éstos deben ser modificables, en caso de que la ventana se "arrastre" o se cambien sus dimensiones, y se deben establecer otros atributos tales como los colores de texto y de fondo (intensidades, si fuera monocromática), caracteres fuente, tamaño, etc.

La activación de otra ventana se detecta como un *evento*, en cuyo caso esta ventana recientemente activada queda superpuesta en la aplicación actual. Puede suspender temporalmente el proceso hasta que se la reactive, pero también puede resultar necesario (con MS-Windows, Concurrent DOS y la nueva versión multitarea de GEM) llevar a cabo su proceso como una tarea de fondo. Cuando se lo vuelva a la vida, el proceso se debe visualizar a sí mismo como la ventana situada en el extremo superior de la pantalla y, significativamente, es responsable de volver a dibujarse a sí misma encima de otras ventanas visualizadas en la pantalla.

Todos los programas, independientemente de lo

triviales que sean, deben contener todo el código para su propio tratamiento de ventanas, incluyendo el registro del contenido actual de la ventana y su posición en relación a cualquier otro dato no visualizado.

Esto a su vez significa una adición de unos 10 Kbytes de código fuente antes de que un programa empiece a hacer algo.

No existe ninguna biblioteca de procedimientos GEM o GSX general a la cual llamar para manejar estas tareas domésticas WIMP. Las casas de software que escriben para GEM están programando en MODULA-2, PASCAL, C, BCPL e incluso ensamblador. Pero a pesar de su pericia esto supone problemas, de modo que los usuarios personales deben tenerlo presente cuando piensen en escribir aplicaciones sencillas GEM en unos pocos cientos de líneas de BASIC. No obstante, si usted sabe programar en cualquiera de los lenguajes que acabamos de mencionar, están apareciendo numerosos productos que prometen proporcionar una interface de programación adecuada a las complejidades del sistema WIMP subyacente.

Juego de herramientas WIMP

La firma TDI, con sede en Bristol, fue una de las primeras que comercializó un juego de herramientas útil y asequible para la programación WIMP. Denominado *MODULA-2/ST*, les ofrece a los usuarios del Atari 520ST la posibilidad de programar seriamente para el GEM en el lenguaje que diseñara Niklaus Wirth para Lilith.

Como ya hemos visto, la empresa británica de software Prospero suministra una biblioteca de rutinas para usar con sus compiladores de gran calidad de PASCAL y FORTRAN, denominada *Prospect*. Ésta proporciona una interface de alto nivel para GSX, la primera implementación de subconjuntos de GKS de Digital Research. El sistema GSX se incorpora en el GEM, actuando como una interface de dispositivo virtual de bajo nivel (el corazón del sistema) y el *Prospect* también estará disponible para GEM.

Sin embargo, si usted está interesado en utilizar el GEM, existen varios sistemas para el usuario serio, incluyendo los siguientes:

- Un Atari 520ST (empaquetado con GEM), MODULA-2 y el Toolkit TDI.
- Un Apricot (o, mejor aun, un RML Nimbus), GEM, un compilador de buena calidad de MODULA-2, PASCAL, BCPL O C.
- Un sistema CP/M-86 o MS-DOS con GSX, Pro-PASCAL y la biblioteca de interface para gráficos Prospero Prospect.
- Un IBM PC, AT o 100% compatible, un ensamblador 8086 o C, y el *GEM programmer's toolkit*, de Digital Research. Éste es útil para personas muy experimentadas que posean un conocimiento profundo de programación de sistemas y muchísimo dinero y tiempo libre.

Para quienes no disponen de tiempo para programar, Digital Research posee un juego de aplicaciones que operan bajo el entorno GEM. *Desktop* es un "administrador de escritorio" para ejecutar otras aplicaciones (incluyendo multitareas, en la nueva versión de GEM), tratamiento de archivos, impresión y trazado con accesorios de escritorio tales como un reloj y una calculadora. *GEM Write* es un procesador de textos similar al WIMP, y *GEM Paint* es un paquete de dibujo para aplicaciones artísticas. Los mismos se pueden adquirir individualmente o bien en forma de paquete incluyendo a los tres, bajo el nombre de GEM Collection. Lamentablemente, los productos DR disponibles sólo se ejecutan en ordenadores IBM PC de 256 Kbytes (o máquinas totalmente compatibles) y los usuarios de otras máquinas habrán de esperar a que sus respectivos fabricantes produzcan sus propias versiones.

Las mil caras del GEM

El GEM contiene dos partes funcionales principales: los *servicios de entorno de aplicaciones* (AES) y la *interface de dispositivo virtual* (VDI). Los componentes del GEM AES incluyen las bibliotecas de subrutinas (que incluyen los caparazones para ventanas, seguimiento del ratón, visualización de mensajes y dibujo de objetos) y un *dispatcher* de multitareas, actualmente limitado a tres pero con una capacidad para realizar hasta 12 tareas en la nueva versión de GEM. El AES también incorpora el buffer de artículos de escritorio (para el reloj, la calculadora, etc.), así como el buffer menú/alerta (el detector de eventos). La GEM VDI es esencialmente el núcleo de "ampliación del sistema gráfico" (GSX) derivado del GKS, pero mejorado con RASTER-OPS y FACES. El primero son las operaciones de bit AND, OR y XOR en bloques fuente y objeto. El segundo almacena los caracteres fuente en archivos que se cargan de forma dinámica. Estas facilidades de nivel más bajo están separadas en dos partes. La primera es el GDOS (*graphics device operating system*: sistema operativo de dispositivo gráfico), que es análogo a una interface OS normal. La segunda parte, los manejadores de dispositivos instalables, dependen de la configuración de hardware y dispositivos periféricos de cada sistema. Todas las funciones gráficas independientes de dispositivos y del anfitrión están contenidas en el GDOS, lo que le confiere independencia del sistema (los detalles específicos para cada dispositivo se encuentran en la sección de manejo separada). El GDOS proporciona un sistema para gráficos estándar y portable que permanece constante, independientemente de los atributos de cualquier estación de trabajo específica. El principal punto de entrada en la VDI es una única subrutina con cinco argumentos:

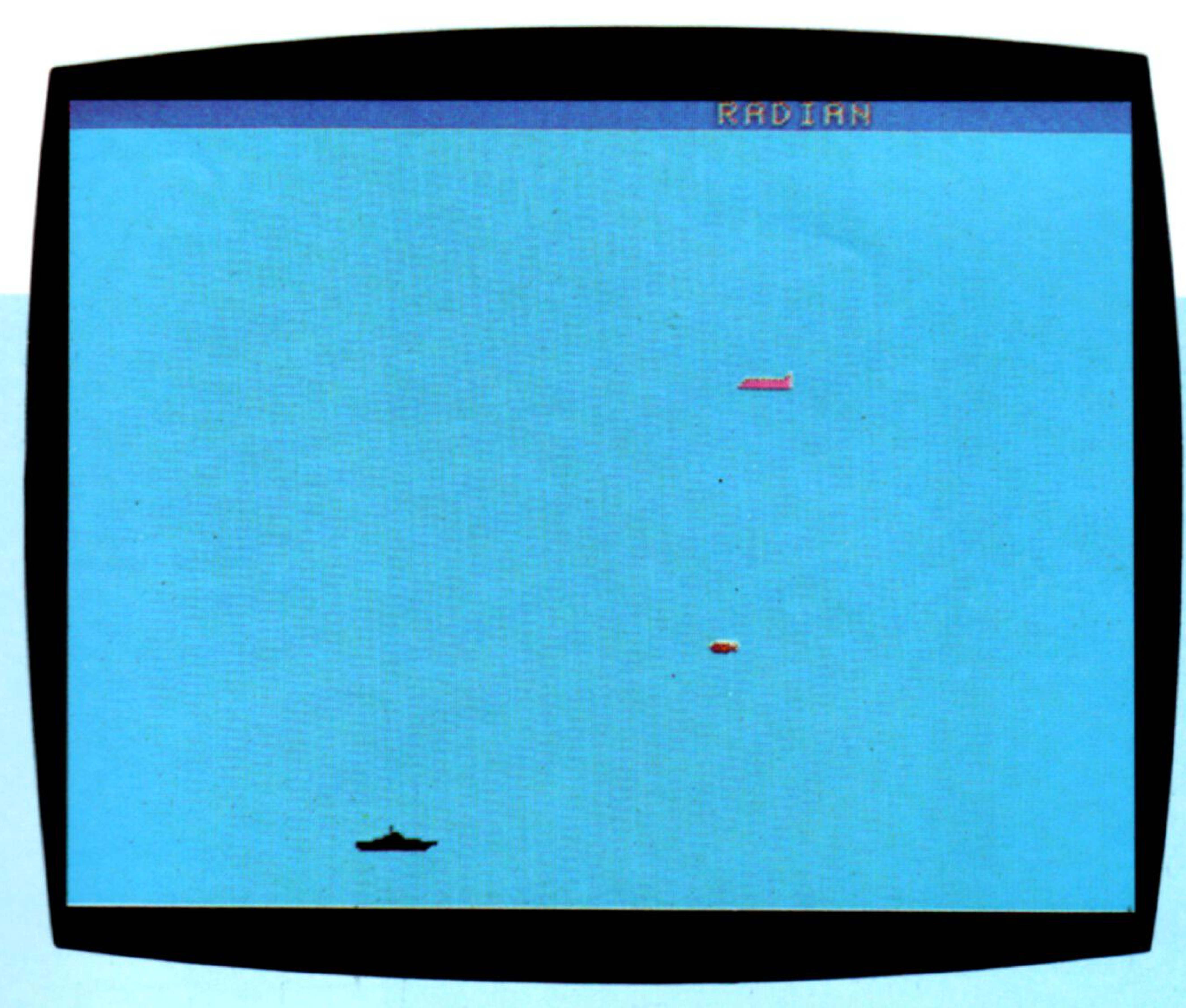
- Matriz de control
- Matriz de parámetros de entrada
- Matriz de coordenadas de punto de entrada
- Matriz de parámetros de salida
- Matriz de coordenadas de punto de salida

Los *bindings* del lenguaje C poseen identificadores que actúan como nombres mnemotécnicos debido a la complejidad y enorme cantidad de llamadas disponibles. De este modo, las llamadas a funciones que, por ejemplo, empiecen con v_ son todas llamadas a rutinas VDI



Exocet en el EXL 100

He aquí de nuevo el mortífero misil Exocet. La versión que presentamos ha sido escrita por Pierre Monsaut para el microordenador EXL 100 de Exelvision



Un portaaviones enemigo se ha aventurado por sus aguas territoriales y hace caso omiso a los requerimientos de identificación. A los mandos de su Mirage 2000, usted debe destruirlo antes de que constituya una amenaza para su base. Para disparar pulse una tecla cualquiera.

```
100 REM **********
110 REM * EXOCET *
120 REM **********
130 R=0
140 GOSUB 970
150 GOSUB 790
160 CALL COLOR("1MC")
170 LOCATE (AY,AX):PRINT A$;
180 IF BX>36 THEN LOCATE (BY,37):PRINT M$;:BB=
    1:GOTO 210
190 CALL COLOR("1BC")
200 LOCATE (BY,BX):PRINT B$;
210 AX=AX-1
220 IF AX<1 THEN LOCATE (AY,1):PRINT M$;:AX=38
230 BB=BB+.2
240 BX=INT (BB)
250 CALL KEY1(D3,D4)
260 IF D3<>255 AND EY=0 THEN
    EX=AX:EY=AY+1:NX=NX-1
270 IF EY<>0 THEN 310
280 FOR I=1 TO 10
290 NEXT I
300 GOTO 160
310 EX=EX-1
320 EY=EY+1
330 LOCATE (EY-1,EX+1):PRINT N$;
340 IF EX<1 THEN EX=38
350 IF EY=23 THEN 400
360 IF EY=22 AND ABS(EX-2-BX)<2 THEN GOSUB
    620
370 CALL COLOR("1RC")
380 LOCATE (EY,EX):PRINT E$;
390 GOTO 160
400 IF EX=40 THEN EX=1
410 LOCATE (EY-1,EX+1):PRINT N$;
420 EY=0
430 EX=0
440 IF NX=0 THEN 460
450 GOTO 160
460 CLS
470 IF S>R THEN R=S
480 CALL KEY1(D3,D4)
490 IF D3<>255 THEN 480
500 CALL COLOR("1BC")
510 LOCATE (10,11)
520 PRINT "PUNTOS:";S;
530 LOCATE (13,11)
540 PRINT "RECORD:";R;
550 LOCATE (16,11)
560 PRINT "OTRA?";
570 CALL KEY1(D3,D4)
580 IF D3=255 THEN 570
590 IF D3<>78 THEN 150
600 CLS
610 END
620 LOCATE (EY-1,EX+1)
630 PRINT N$;
640 S=S+10
650 LOCATE (EY,EX)
660 CALL COLOR("1bC")
670 PRINT F$;
680 FOR I=1 TO 30
690 X=INTRND(4)-2
700 Y=INTRND(6)-1
710 LOCATE (EY-Y,EX-X)
720 PRINT F$;
730 NEXT I
740 FOR I=1 TO 200
750 NEXT I
760 NX=NX+1
770 CLS
780 GOTO 160
790 CLS
800 B$=CHR$(32)&CHR$(100)&CHR$(101)&CHR$(102)
810 AX=38
820 S=0
830 BB=1
840 BX=1
850 A$=CHR$(103)&CHR$(104)&CHR$(32)
860 N$=CHR$(32)
870 MS=N$&N$&N$
880 E$=CHR$(105)
890 F$=CHR$(106)
900 EX=0
910 EY=0
920 XC=2
930 NX=20
940 BY=22
950 AY=B
960 RETURN
970 CLS ("BCb")
980 CALL CHAR(100,"000000000007FFFF7F00")
990 CALL CHAR(101,"00202038FCFFFFFFFF00")
1000 CALL CHAR(102,"0000000000E0FFFCF800")
1010 CALL CHAR(103,"0000000000003F7FFF00")
1020 CALL CHAR(104,"000000000103FFFFFF00")
1030 CALL CHAR(105,"00000000007DFF7D0000")
1040 CALL CHAR(106,"000B21800A0028001000")
1050 RETURN
```

¡Cartas arriba!

Proseguimos nuestro proyecto dedicado al veintiuno ocupándonos de las rutinas de visualización de naipes y barajada del mazo para Amstrad, Spectrum y BBC. Necesitaremos definir caracteres para los palos de las cartas y los bordes, además de resolver un problema que se presenta con la función TAB en estos micros. La tabulación a una cierta posición de una línea borra todos los datos a la izquierda de esa posición TAB. Esto exige adaptar ligeramente la rutina de visualización.

Visualización de naipes y barajada

Gama Amstrad CPC

Programa principal:

```
10 REM **** Veintiuno Amstrad ****
20 GOSUB 500:REM inic matriz etc
50 REM **** Bucle del juego ****
55 GOSUB 600: REM inic juego
```

Inicialización de matrices:

```
500 REM **** inic matrices etc ****
505 INK 0,16:rosa=0:INK 1,3:rojo=1:INK
    2,0:negro=2:INK 3,26:blanco=3
512 bk$=STRING$(7,207)
513 br$=CHR$(149):li$=STRING$(7,154)
515 DIM x(2),y(2):REM siguientes posicion naipes
520 su$=CHR$(228)+CHR$(227)+CHR$(226)+CHR$(229):REM
    simbolos palos
530 DIM cn$(13):FOR i=1 TO 13:READ cn$(i):NEXT i:REM leer
    datos numero
540 DIM cd$(13):FOR i=1 TO 13:READ cd$(i):NEXT i:REM leer
    datos de patron
560 DIM dk(52,2):REM baraja naipes
570 GOSUB 3000:REM barajar mazo
580 BORDER 9:PAPER rosa:REM colores pantalla
590 RETURN
600 REM **** inic juego ****
605 CLS
610 hp(1)=1:hp(2)=1
620 x(1)=0:y(1)=0:x(2)=20:y(2)=0
625 bt=0:sb=0:GOSUB 4200:REM imprimir apuestas
630 RETURN
```

Rutina de visualización de naipes:

```
900 REM **** impresión en ****
910 LOCATE tx+1,ty+1:RETURN
1000 REM **** visualizar naipe ****
1010 GOSUB 1050:GOSUB 1100:RETURN
1050 REM **** naipe en blanco ****
1055 tx=x(pl):ty=y(pl):GOSUB 900: REM posicion
1060 PEN blanco:PRINT CHR$(150);li$:CHR$(156)
1070 FOR i=1 TO 9:tx=x(pl):ty=ty+1:GOSUB 900:PRINT
     br$;SPACE$(7);br$:NEXT i
1080 tx=x(pl):ty=ty+1:GOSUB 900:PRINT
     CHR$(147);li$;CHR$(153)
1090 RETURN
1100 REM **** detalles naipe ****
1120 tx=x(pl)+2:ty=y(pl)+1:GOSUB 900:REM posicion
1125 ct$=MID$(su$,su,1):REM seleccionar tipo de palo
1127 co=rojo:IF su> 2 THEN co=negro
1128 PEN co
1130 FOR i=1 TO 19 STEP 3
1140 cc$=MID$(cd$(cn),i,3):cl$=""
1142 FOR j=1 TO 3:c$=SPACE$(2)
1144 IF MID$(cc$,j,1)="1" THEN c$=ct$+SPACE$(1)
1146 cl$=cl$+c$:NEXT j
1150 tx=x(pl)+2:ty=ty+1:GOSUB 900:PRINT
     cl$:NEXT i
1160 REM **** añadir etiquetas angulos ****
1170 tx=x(pl)+1:ty=y(pl)+1:GOSUB 900: PRINT
     cn$(cn):REM numero
1180 ty=ty+1:GOSUB 900:PRINT ct$:REM palo
1190 tx=x(pl)+7:ty=y(pl)+9:GOSUB 900:PRINT cn$(cn):REM
     numero de abajo
1192 x(pl)=x(pl)+2:y(pl)= y (pl)+1
1195 RETURN
1200 REM **** visualizar reverso naipe ****
1210 GOSUB 1050:GOSUB 1250:RETURN
1250 REM **** reverso naipe ****
1255 tx=x(pl):ty=y(pl):GOSUB 900:REM posicion
1260 FOR i=1 TO 9:tx=x(pl):ty=ty+1:GOSUB 900:PRINT
     br$;:PEN rojo:PRINT bk$;:PEN blanco:PRINT br$:NEXT i
1270 x(pl)=x(pl)+2:y(pl)=y(pl)+1
1280 RETURN
1300 REM **** repartir un naipe ****
1310 cn=dk(dp,1):su=dk(dp,2)
1320 dp=dp+1:IF dp> 52 THEN dp=1
1330 IF fl=1 THEN GOSUB 1200:RETURN:REM visualizar reservo
     naipe
1335 GOSUB 1000:RETURN:REM visualizar naipe
2000 REM **** datos numeros naipe ****
2010 DATA A,2,3,4,5,6,7,8,9,T,J,Q,K
2100 REM **** datos visualizacion naipes ****
2110 DATA "000000000010000000000"
2120 DATA "000010000000000010000"
2130 DATA "000010000010000010000"
2140 DATA "000101000000000101000"
2150 DATA "000101000010000101000"
2160 DATA "000101000101000101000"
2170 DATA "000101010101000101000"
2180 DATA "000101010101010101000"
2190 DATA "101000101010101000101"
2200 DATA "101010101000101010101"
2210 DATA "000000000010000000000"
2220 DATA "000000000010000000000"
2230 DATA "000000000010000000000"
```

Mezclando la baraja:

```
3000 REM **** barajar el mazo ****
3005 RANDOMIZE TIME:dp=1
3007 FOR i=1 TO 52:dk(i,1)=0:NEXT i
3010 FOR i=1 TO 4:FOR j=1 TO 13
3020 ep=INT(RND(1)* 52)+1:REM seleccionar punto entrada
3030 IF dk(ep,1)=0 THEN 3050
3040 ep=ep+1:IF ep>52 THEN ep=1
3050 GOTO 3030
3050 dk(ep,1)=j:dk(ep,2)=i:NEXT j,i
3060 RETURN
```

BBC MICRO

Nota: Digite PAGE=$0E00 antes de entrar el programa o cargarlo desde disco o cinta

Programa principal:

```
10 REM VEINTIUNO BBC
15 MODE 1
20 GOSUB 500
50 REM
55 GOSUB 600
```

Inicialización de matrices:

```
500 REM
510 SP$="":FOR I=1 TO 39:SP$=SP$+"":NEXT I
512 BK$="":LI$="":FOR I=1 TO
    7:LI$=LI$+CHR$(195):BK$=BK$+CHR$(166): NEXT I
```

```
513 BR$=CHR$(194)
515 DIM X(2),Y(2)
520 SU$=CHR$(211)+CHR$(218)+CHR$(216)+
    CHR$(193)
530 DIM CN$(13):FOR I=1 TO 13:READ CN$(I):
    NEXT
535 CN$(10)=CHR$(128)
540 DIM CD$(13):FOR I=1 TO 13:READ CD$(I):
    NEXT
560 DIM DK(52,2)
570 GOSUB 3000
571 COLOUR 131:COLOUR 0
572 VDU 23,166,165,66,90,36,90,165,90,66
573 VDU 23,195,0,0,0,255,255,0,0,0
574 VDU 23,194,24,24,24,24,24,24,24,24
575 VDU 23,193,16,56,124,254,254,238,84,56
576 VDU 23,216,24,60,90,255,255,90,24,60
577 VDU 23,218,16,56,124,254,254,124,56,16
578 VDU 23,211,36,126,255,255,255,126,60,24
579 VDU 23,213,0,0,0,15,31,28,24,24
580 VDU 23,201,0,0,0,240,248,56,24,24
581 VDU 23,202,24,24,28,31,15,0,0,0
582 VDU 23,203,24,24,56,248,240,0,0,0
583 VDU 23,128,206,219,219,219,219,219,206,0
590 RETURN
600 REM
601 DB=0:CS=0
605 CLS
620 X(1)=0:Y(1)=0:X(2)=20:Y(2)=0
630 RETURN
```

Rutina de visualización de naipes:

```
900 REM **** IMPRIMIR EN ****
910 PRINT TAB(TX,TY);:RETURN
1000 REM
1010 GOSUB 1050:GOSUB 1100:RETURN
1050 REM
1055 TX=X(PL):TY=Y(PL):GOSUB 900:REM POSICION
1060 COLOUR 0:PRINT TAB(TX,TY);CHR$(213);LI$;
     CHR$(201)
1070 FOR I=1 TO 9:PRINT TAB(TX,TY+I);BR$" ";BR$:
     NEXT I
1080 PRINT TAB(TX,TY+10);CHR$(202);LI$;CHR$(203)
1090 RETURN
1100 REM
1120 TX=X(PL)+2:TY=Y(PL)+1:GOSUB 900
1125 CT$=MID$(SU$,SU,1)
1127 COLOUR 1:IF SU>2 THEN COLOUR 0
1130 FOR I=1 TO 19 STEP 3
1140 CC$=MID$(CD$(CN),I,3):CL$=""
1142 FOR J=1 TO 3:C$=CHR$(9)+CHR$(9)
1144 IF MID$(CC$,J,I)="1" THEN C$=CT$+CHR$(9)
1146 CL$=CL$+C$:NEXT J
1150 PRINT TAB(X(PL)+2,TY+((I+3)/3));CL$:NEXT I
1170 TX=X(PL)+1:TY=Y(PL)+1:GOSUB 900:PRINT
     CN$(CN)
1180 TY=TY+1:GOSUB 900:PRINT CT$
1190 TX=X(PL)+7:TY=Y(PL)+9:GOSUB 900:PRINT
     CN$(CN)
1192 X(PL)=X(PL)+2:Y(PL)=Y(PL)+1
1195 RETURN
1200 REM
1210 GOSUB 1050:GOSUB 1250:RETURN
1250 REM
1255 TX=X(PL):TY=Y(PL)+1:GOSUB 900
1260 FOR I=1 TO 9:PRINT TAB(TX,I);BR$;BK$;BR$:
     NEXT I
1270 X(PL)=X(PL)+2:Y(PL)=Y(PL)+1
1280 RETURN
1300 REM
1310 CN=DK(DP,1):SU=DK(DP,2)
1320 DP=DP+1:IF DP>52 THEN DP=1
1330 IF FL=1 THEN GOSUB 1200:RETURN
1335 GOSUB 1000:RETURN
2010 DATA A,2,3,4,5,6,7,8,9,T,J,Q,K
2110 DATA"000000000010000000000"
2120 DATA"000010000000000010000"
2130 DATA"000010000010000010000"
2140 DATA"000101000000000101000"
2150 DATA"000101000010000101000"
2160 DATA"000101000101000101000"
2170 DATA"000101010101000101000"
2180 DATA"000101010101010101000"
2190 DATA"101000101010101000101"
2200 DATA"101010101000101010101"
2210 DATA"000000000010000000000":REM J
2220 DATA"000000000010000000000":REM Q
2230 DATA"000000000010000000000":REM K
```

Mezclando la baraja:

```
3000 REM
3005 R=RND(-TIME):DP=1
3007 FOR I=1 TO 52:DK(I,1)=0:NEXT I
3010 FOR I=1 TO 4:FOR J=1 TO 13
3020 EP=INT(RND(1)*52)+1
3030 IF DK(EP,1)=0 THEN 3050
3040 EP=EP+1:IF EP>52 THEN EP=1
3045 GOTO 3030
3050 DK(EP,1)=J:DK(EP,2)=I:NEXT J,I
3060 RETURN
```

Sinclair Spectrum:

Programa principal:

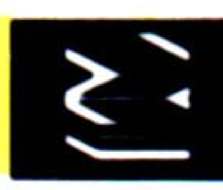
```
10>REM **** VEINTIUNO SPECTRUM ****
15 PRINT "ESPERE POR FAVOR..."
16 POKE 23658,8
20 GO SUB 500: REM INIC MATRICES ETC
50 REM **** BUCLE DEL JUEGO ****
55 GO SUB 600:REM INIC JUEGO
```

Inicialización de matrices:

```
500>REM **** INIC MATRICES ETC ****
510 LET S$="":FOR I=1 TO 25: LET S$=S$+"":
    NEXT I
511 LET S$=S$+S$(TO 14)
512 LET B$="": LET L$="": FOR I=1 TO 7: LET L$=L$+
    CHR$ 145: LET B$= B$+CHR$ 144:NEXT I
513 DET D$=CHR$ 146
515 DIM X(2): DIM Y (2): REM SIGUIENTES POSICIONES
    NAIPES
520 LET U$=CHR$ 147+CHR$ 148+CHR$ 149+CHR$ 150:
    REM TIPOS DE PALO
530 DIM N$(13): FOR I=1 TO 13: READ N$(I):NEXT I:REM LEER
    DATOS NUMEROS
540 DIM C$(13,21): FOR I=1 TO 13: READ C$(I): NEXT I: REM
    LEER DATOS PATRON
560 DIM K(52,2): REM PREPARAR MATRIZ BARAJA
    NAIPES
570 GO SUB 3000: REM BARAJAR MAZO
580 BORDER 4: PAPER 7: INK 9: CLS
581 REM
582 FOR L=USR "A" TO USR "L"+7
583 READ US: POKE L,US: NEXT L
590 RETURN
600 REM **** INIC JUEGO ****
602 LET DB=0: LET CS=0
605 CLS
620 LET X(1)=0: LET Y(1)=0: LET X(2)=14: LET Y(2)=0
630 RETURN
```

Rutina de visualización de naipes:

```
900 >REM **** IMPRIMIR EN ****
910  PRINT AT TY,TX;:RETURN
1000 REM **** VISUALIZAR NAIPE ****
1010 GO SUB 1050: GO SUB 1100: RETURN
1050 REM **** NAIPE EN BLANCO ****
1055 LET TX=X(PL)+1:LET TY=Y(PL): GO SUB 900: REM POSI-
     CION
1060 PRINT AT Y(PL),TX;CHR$ 151;L$;CHR$ 152
1070 FOR I=1 TO 9: PRINT AT Y(PL)+I,TX:CHR$ 146;" ";CHR$
     146: NEXT I
1080 PRINT AT 10+Y(PL),TX;CHR$ 153;L$;CHR$ 154
1090 RETURN
1100 REM **** DETALLES NAIPE ****
1120 LET TX=X(PL)+2: LET TY=Y(PL)+2:GO SUB 900: REM
     POSICION
1125 LET T$=U$(SU): REM SELECCIONAR TIPO DE
     PALO
```



```
1127 INK 0: IF SU>2 THEN INK 2:REM SELECCIONAR
     COLOR
1130 FOR I=1 TO 19 STEP 3
1140 LET Z$=C$(CN)(I TO I+2)
1142 FOR J=1 TO 3: LET W$=" "
1143 PRINT " ";
1144 IF Z$(J)="1" THEN PRINT T$;
1145 IF Z$(J)="0" THEN PRINT " ";
1146 NEXT J
1150 PRINT AT ((I+3)/3)+TY,X(PL)+2;:NEXT I
1160 REM **** AÑADIR ETIQUETAS ANGULOS ****
1170 LET TX=X(PL)+2: LET TY=Y(PL)+1: GO SUB 900: PRINT
     N$(CN): REM POSICION
1180 LET TY=TY+1: GO SUB 900: PRINT T$: REM
     PALO
1190 LET TX=X(PL)+8: LET TY=Y(PL)+9: GO SUB 900: PRINT
     N$(CN): REM POSICION
1192 LET X(PL)=X(PL)+2: LET Y(PL)=Y(PL)+1
1193 INK 0
1195 RETURN
1200 REM **** VISUALIZAR REVERSO NAIPE ****
1210 GO SUB 1050: INK 1: GOSUB 1250: INK 0:
     RETURN
1250 REM **** REVERSO NAIPE ****
1255 LET TX=X(PL)+2: LET TY=Y(PL)+1: GO SUB 900: REM
     POSICION
1260 FOR I=1 TO 9: PRINT AT I,TX;B$: NEXT I
1270 LET X(PL)=X(PL) +2: LET Y (PL)=Y(PL)+1
1280 RETURN
1300 REM **** REPARTIR UN NAIPE ****
1310 LET CN=K(DP,1): LET SU=K(DP,2)
1320 LET DP=DP+1: IF DP>52 THEN LET DP=1
1330 IF FL=1 THEN GO SUB 1200: RETURN: REM VISUALIZAR
     REVERSO NAIPE
1335 GO SUB 1000: RETURN: REM VISUALIZAR
     NAIPE
2000 REM **** DATOS NUMEROS DE NAIPES ****
2010 DATA "A","2","3","4","5","6","7","8","9",CHR$ 155,
     "J","Q","K"
2100 REM **** DATOS VISUALIZACION NAIPES ****
2110 DATA "000000000010000000000": REM A
2120 DATA "000010000000000010000": REM 2
2130 DATA "000010000010000010000": REM 3
2140 DATA "000101000000000101000": REM 4
2150 DATA "000101000010000101000": REM 5
2160 DATA "000101000101000101000": REM 6
2170 DATA "000101010101000101000": REM 7
2180 DATA "000101010101010101000": REM 8
2190 DATA "101000101010101000101": REM 9
2200 DATA "101010101000101010101": REM 10
2210 DATA "000000000010000000000": REM J
2220 DATA "000000000010000000000": REM Q
2230 DATA "000000000010000000000": REM K
2240 REM UDG
2250 DATA 165,66,90,36,90,165,90,66
2251 DATA 0,0,0,255,255,0,0,0
2252 DATA 24,24,24,24,24,24,24,24,
2253 DATA 16,56,124,254,254,238,84,56
2254 DATA 24,60,90,255,255,90,24,60
2255 DATA 16,56,124,254,254,124,56,16
2256 DATA 36,126,255,255,255,126,60,24
2257 DATA 0,0,0,15,31,28,24,24
2258 DATA 0,0,0,240,248,56,24,24
2259 DATA 24,24,28,31,15,0,0,0
2260 DATA 24,24,56,248,240,0,0,0
2261 DATA 0,76,82,82,82,82,76,0
```

Mezclando la baraja:

```
3000 >REM **** BARAJANDO EL MAZO ****
3005 RANDOMIZE: LET DP=1
3007 FOR I=1 TO 52: LET K(I,1)=0: NEXT I
3010 FOR I=1 TO 4: FOR J=1 TO 13
3020 LET EP= INT (RND*52)+: REM SELECCIONAR PUNTO
     ENTRADA
3030 IF K(EP,1)=0 THEN GO TO 3050
3040 LET EP=EP+1: IF EP>52 THEN LET EP=1
3045 GO TO 3030
3050 LET K(EP,1)=J: LET K(EP,2)=I: NEXT J:
     NEXT I
3060 RETURN
```

En el borde
Todas estas versiones del veintiuno utilizan gráficos de caracteres para dibujar el borde de los naipes. En las versiones para el BBC Micro y el Spectrum hemos de definir nosotros los caracteres de los bordes. En la versión para el BBC Micro, todas las definiciones de palo y borde se realizan utilizando la instrucción VDU 23 entre 572-583. En la versión para el Spectrum, las definiciones de caracteres se retienen como DATA entre 2240-2261 y el READ forma parte de la subrutina de inicialización entre 581-583

Superior izquierdo: 0, 0, 0, 15, 31, 28, 24, 24
Línea horizontal: 0, 0, 0, 255, 255, 0, 0, 0
Superior derecho: 0, 0, 0, 240, 240, 56, 24, 24
Símbolo 10: 0, 76, 82, 82, 82, 82, 76, 0
Picas: 16, 56, 124, 254, 254, 238, 84, 86
Línea vertical: 24, 24, 24, 24, 24, 24, 24, 24
Corazones: 36, 126, 255, 255, 255, 126, 60, 24
Diamantes: 16, 56, 124, 254, 254, 124, 56, 56
Reverso del naipe: 66, 90, 36, 90, 165, 90, 66
Línea vertical: 24, 24, 24, 24, 24, 24, 24, 24
Tréboles: 24, 60, 90, 255, 255, 90, 24, 60
Símbolo 10: 0, 76, 82, 82, 82, 82, 76, 0
Inferior izquierdo: 24, 24, 28, 31, 15, 0, 0, 0
Horizontal: 0, 0, 0, 240, 240, 56, 24, 24
Inferior derecho: 24, 24, 56, 240, 240, 0, 0, 0

Caroline Clayton

Palos a medida
Si bien el Commodore 64 y la gama de micros Amstrad CPC poseen caracteres de palos de baraja como parte de sus juegos de caracteres estándares, en la versiones para el BBC Micro y el Spectrum necesitamos definir nuestros caracteres de palos

Ir y venir

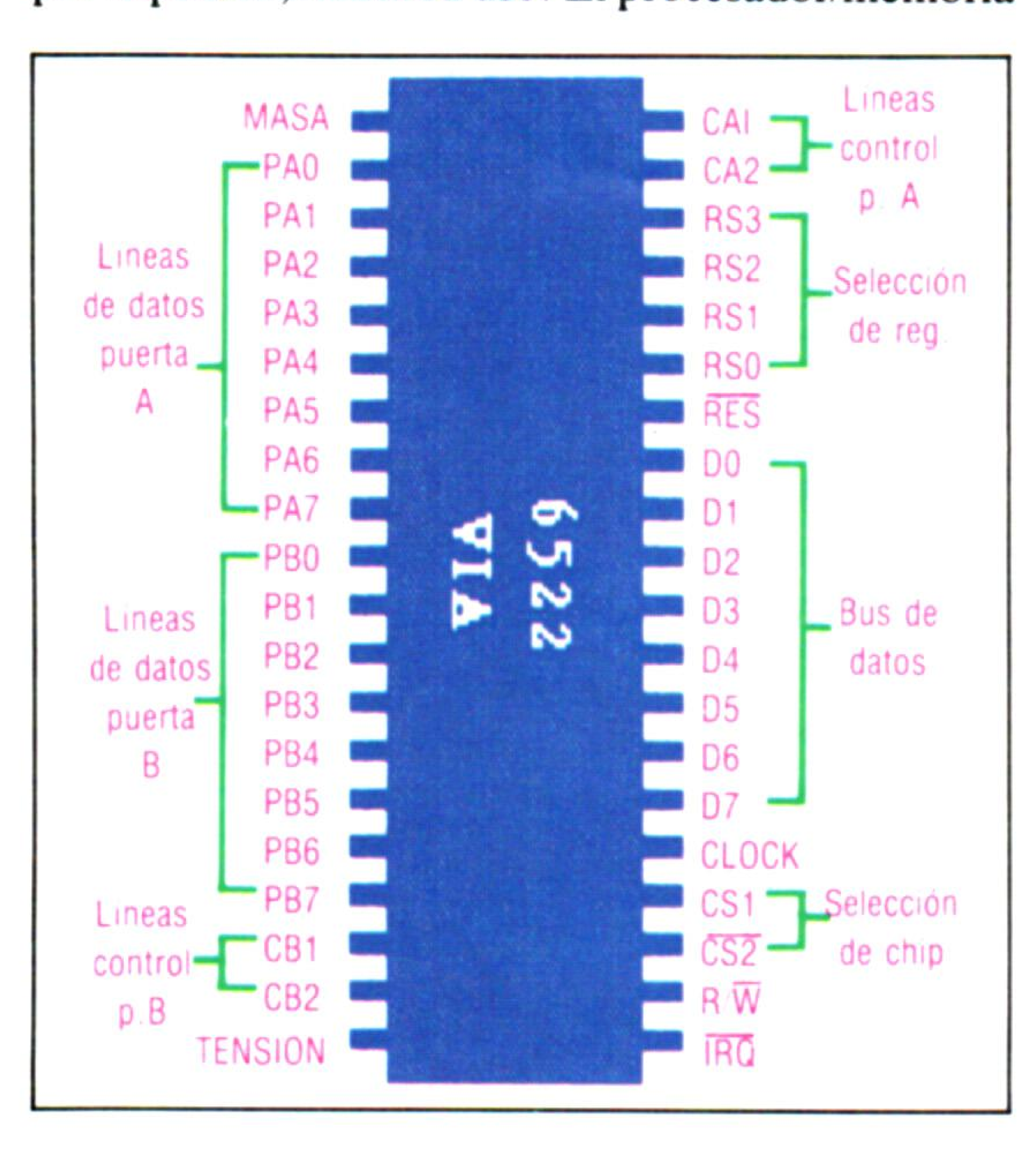
Centraremos nuestra atención en las facilidades de E/S en serie y en paralelo

Hasta ahora hemos analizado el procesador y cómo se comunica con la memoria ROM y RAM. El sistema que hemos descrito es autocontenido, lo que, por supuesto, limita su uso. El procesador/memoria básico se comunica con otros dispositivos de muchas maneras. Las entradas se pueden obtener utilizando un teclado, una palanca de mando o a través de algún dispositivo externo unido al sistema a través de una puerta, y la salida normalmente se efectúa a través de una pantalla o una impresora.

No podemos conectar directamente dispositivos de E/S al bus de datos principal porque los datos de salida normalmente no están presentes en el mismo el tiempo suficiente como para que un dispositivo externo los utilice. Por el contrario, los datos que entran se deben retener en el bus de datos el tiempo suficiente para que el procesador se ocupe de ellos. En consecuencia, a nivel general la tarea de una interface de E/S consiste en congelar los datos el tiempo suficiente para que se realice la entrada o salida y para sincronizar las operaciones E/S.

Por supuesto, la mayoría de los dispositivos de E/S hacen muchas más cosas además de proporcionar un par de *latches* y líneas de control. Pero antes de seguir adelante, vale la pena hacer la distinción entre los dos tipos principales de dispositivo de E/S con que cuentan la mayoría de los micros. Si ignoramos el problema de la salida a la pantalla, que por lo general se maneja a través de sistemas de circuitos especializados (debido, entre otras cosas, a la velocidad de transferencia de datos requerida para actualizar una pantalla), el tipo de interface E/S utilizada suele depender del método en que estén unidos los dispositivos, que será un enlace ya sea en serie o bien en paralelo. Por lo tanto, los dispositivos de E/S se dividen en dos categorías principales: dispositivos PIO y dispositivos SIO.

Ya hemos visto de forma detallada métodos de comunicaciones en serie. En particular, nuestra serie sobre la interface MIDI en el apartado *Bricolaje* proporciona un ejemplo clásico de cómo imple-



Patillas in/out
El chip VIA 6522 utilizado en el BBC Micro constituye un ejemplo típico de chip de E/S en paralelo: posee dos puertas de ocho bits, y líneas al bus de datos

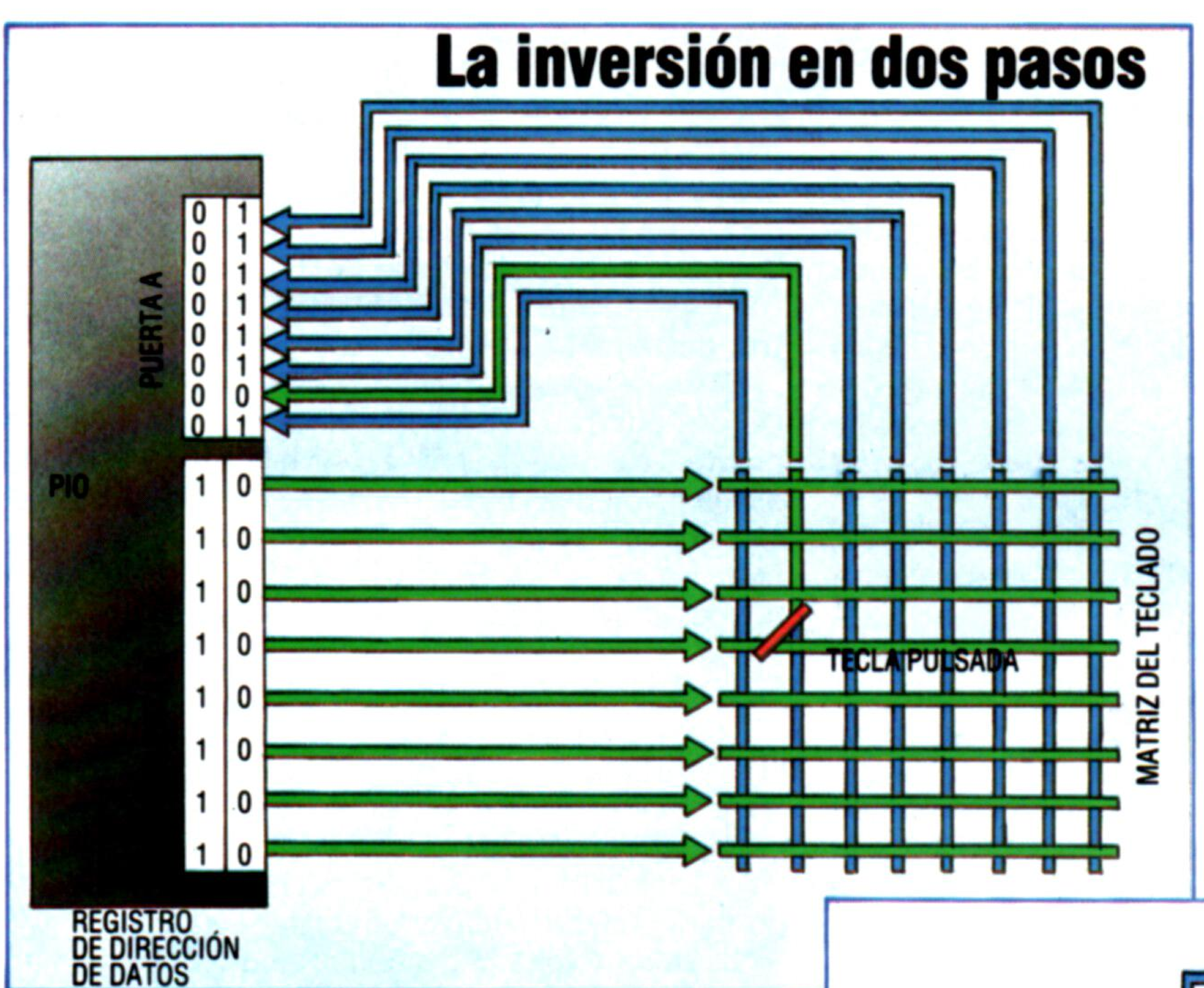
El código PIO
Los chips PIO se pueden programar colocando valores en sus registros internos. Para permitir el acceso a estos registros, el procesador ha de ser capaz de "verlos" en su propio espacio de direcciones. Este ejemplo muestra dos chips PIO preparados para que su contenido aparezca en $DC00 y $DD00. Se decodifican los siete bits superiores del bus de direcciones y A8 selecciona $DC00 o $DD00. Puesto que cada PIO posee 16 registros internos, los cuatro bits inferiores del bus de direcciones se utilizan para direccionar un registro individual

A15
A13
A11
A9
A7
A5
A3
A1
A0
VIA 2
VIA 1
CS1
CS2
R/W

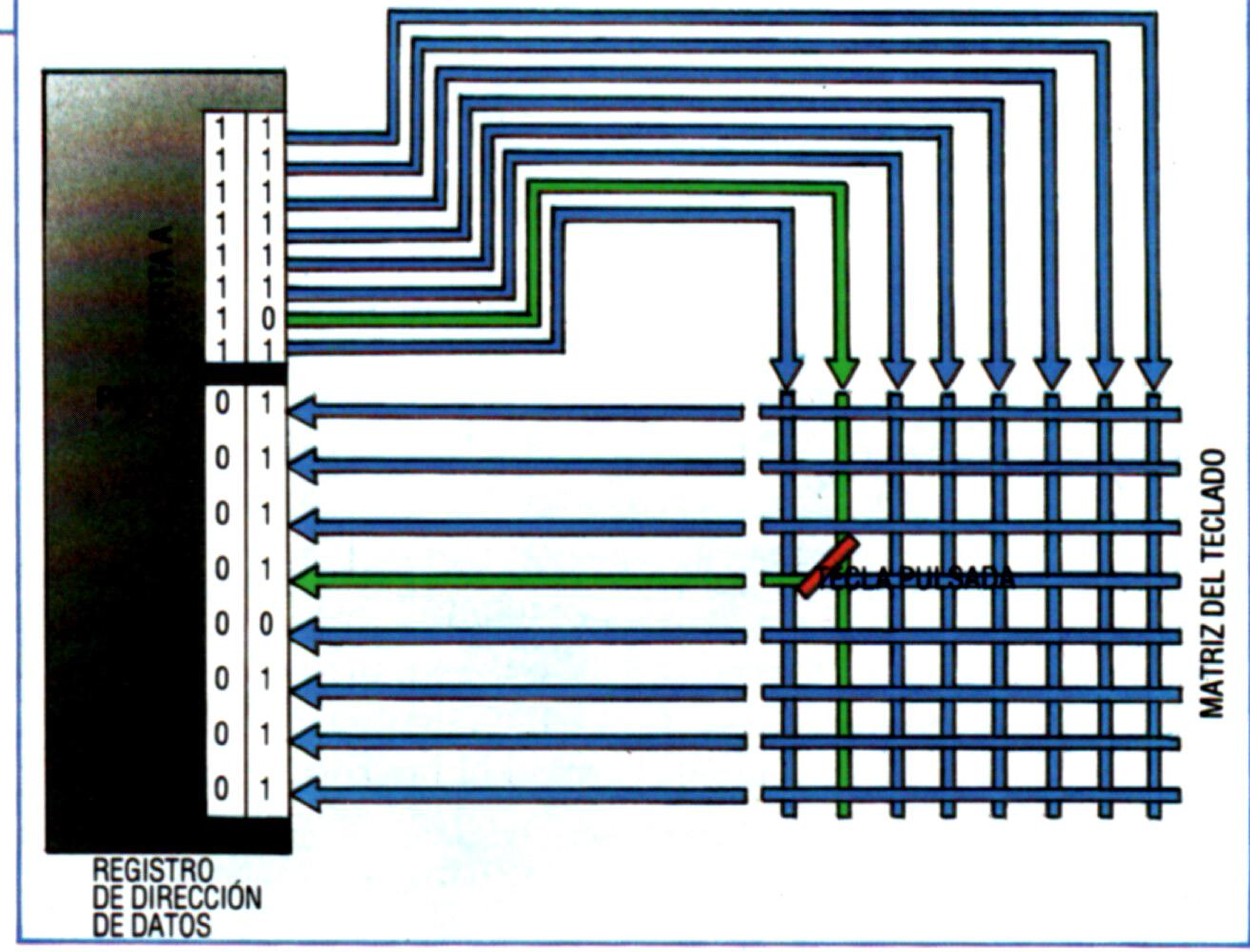
Líneas de direcciones PIO

Kevin Jones





La clave del problema
La técnica de inversión de líneas es uno de los diversos métodos que se utilizan para identificar pulsaciones de teclas en el teclado, y emplea la programabilidad direccional de las dos puertas PIO de ocho bits para producir ceros a lo largo de las filas de la matriz del teclado y producir luego el resultado recibido en la otra puerta a lo largo de las columnas. El código de 16 bits resultante retenido en los registros de datos de las dos puertas identifica inequívocamente la tecla pulsada



mentar un dispositivo de comunicaciones asíncronas. Los componentes esenciales de una interface de ese tipo son registros de desplazamiento, que permiten convertir los bits de datos en paralelo provenientes del bus de datos en un flujo en serie, y la conversión de un flujo de entrada a un formato en paralelo. Este dispositivo suele manejar la generación y comprobación de paridad automática.

Para todo sistema de ordenador personal, el chip PIO reviste una importancia más inmediata. Sin embargo, el término PIO no se utiliza con carácter universal, y términos tales como VIA (*versatile interface adaptor*: adaptador versátil de interface), PIA (*peripheral interface adaptor*: adaptador de interface para periféricos) y CIA (*complex interface adaptor*: adaptador de interface compleja) se refieren esencialmente al mismo dispositivo.

Los micros personales con facilidades completas para conexión en interface, tales como el BBC Micro y el Commodore 64, están equipados con dos chips PIO, uno de los cuales normalmente está dedicado a los dispositivos de E/S incorporados, como el teclado y las puertas para palanca de mando. El segundo PIO se puede emplear conjuntamente con una impresora en paralelo y una puerta para el usuario, dando al usuario o a los fabricantes un acceso independiente al sistema de E/S.

En el diagrama (página contigua, arriba) vemos el chip VIA 6522 utilizado en el BBC Micro. Disponiendo de 40 patillas, este paquete proporciona ocho enlaces de bus de datos y dos puertas de ocho bits que se pueden utilizar para entrada o salida. Otras líneas incluyen controles de interrupciones y de lectura/escritura, bits de selección de registro y líneas de comunicación para las dos puertas E/S.

Aunque un chip PIO es, desde el punto de vista electrónico, un dispositivo sumamente complejo, resulta bastante sencillo comprenderlo en su aspecto de funcionamiento. Típicamente, un chip PIO incluido en un micro de ocho bits tiene dos puertas de E/S (A y B) con control de dirección de datos individual en cada bit de puerta proporcionado por registros de dirección de datos y líneas de control de comunicación separadas. Muchos PIO incluyen también un registro de desplazamiento que se puede utilizar para transmitir o recibir datos a lo largo de una de las líneas de puerta de comunicación. Éstas suelen estar bajo el control de temporización interna de uno de los dos temporizadores, o bajo una fuente temporizadora externa entrada a través de otra línea de comunicación.

Un registro de estado de interrupciones de ocho bits utiliza cada bit para indicar una solicitud de interrupción desde un cierto número de fuentes: las líneas de control de comunicación, los temporizadores o los registros de desplazamiento. Un registro de permisión de interrupciones programable permite que el programador seleccione cuál (si la hubiera) de las fuentes de interrupciones PIO necesita realmente interrumpir al procesador.

Concentrémonos ahora en cómo un PIO se puede conectar al sistema procesador/memoria. La mayoría de los sistemas con facilidades de E/S com-

pletas emplean dos chips PIO. Supongamos que quisiéramos que los registros internos de estos dos dispositivos aparecieran en $DC00 y $DD00. Para pasar datos a los PIO y recibir datos provenientes de ellos, el procesador debe ser capaz de tratar los registros internos del PIO como si fueran posiciones de memoria y direccionarlos de la misma forma. Para hacer que los registros PIO aparezcan en el espacio de direcciones, debemos, por lo tanto, manipular algunas líneas del bus de direcciones.

El diagrama ilustra cómo se tratan los ocho bits superiores del bus de direcciones para seleccionar cualquiera de los PIO. Los cuatro bits superiores se operan para producir un uno cuando 1101 ($D) está presente. Los tres bits siguientes se operan para producir un uno cuando 110 está presente. La selección entre $DC0 y $DD0 se realiza realmente mediante el bit de dirección A8. Los cuatro bits inferiores conectan directamente con las cuatro patillas de selección de registro del PIO. Por tanto, los registros internos del PIO 1 se pueden direccionar como las posiciones de $DC00 a $DC0F, y los del PIO 2 se pueden direccionar como de $DD00 a $DD0F.

Normalmente un PIO es exclusivo del sistema, para la conexión de palancas de mando y el teclado del ordenador principal. Existen varios métodos de conectar un teclado complejo de 64 teclas, el más simple de los cuales consiste en construir el teclado de modo que tenga por debajo una matriz de cables, dispuestos en ocho filas y ocho columnas. Entonces se conectan las líneas de fila a una puerta PIO y las líneas de columna a otra, como se indica en el diagrama. Cuando se pulsa una tecla, se conectan los cables de la fila y columna correspondientes a la intersección.

El método de inversión de línea utilizado para detectar qué tecla se ha pulsado se basa fundamentalmente en software, dependiendo de la capacidad que posee la puerta del PIO para programarse para entrada o salida. El paso 1 supone establecer la puerta B en salida y colocar cero en su registro de datos. La puerta A se establece para entrada y, dado que las líneas de columna están todas retenidas *high*, el bit del registro de datos de la puerta B correspondiente a la tecla pulsada estará *low*, mientras que todos los otros estarán *high*.

El paso 2 implica invertir la dirección de datos de las dos puertas. Ahora la puerta B actúa como la puerta de entrada y cada bit se establecerá en uno, a excepción del bit que corresponde a la fila de teclas que se esté pulsando. El código de 16 bits retenido ahora en los registros de las dos puertas se puede utilizar para identificar la posición de la ROM de caracteres que contiene el código de ocho bits del carácter cuya tecla se ha pulsado.

Otro de los métodos para identificación de tecla es la exploración de filas. En este caso cada columna se establece *high*, buscando a la vez un correspondiente nivel *high* en una de las líneas de fila y chips exclusivos que decodifican la matriz del teclado para generar códigos directamente.

Ya hemos visto la decodificación de direcciones PIO. El diagrama final (abajo) muestra cómo se puede utilizar un par de chips PIO para acceder al teclado y proporcionar puertas para el usuario e impresora en paralelo. Mientras que el primero puede depender completamente de la conexión al teclado, el segundo PIO dispone de dos puertas de ocho bits. La primera de ellas se puede utilizar como una puerta de salida para una impresora, a través de un buffer adecuado para proteger al PIO contra conexiones incorrectas en la puerta para impresora. La segunda queda, entonces, disponible como puerta para el usuario.

Centinelas gemelos
Este diagrama muestra cómo se pueden utilizar dos chips PIO para conectar en interface un teclado, impresora en paralelo, palanca de mando y puerta para el usuario al bus de datos. Una de las principales tareas del firmware o las rutinas kernel que se proporcionan con el sistema es programar los PIO de modo que manipulen los datos entrantes y salientes. Por consiguiente, para la mayoría de las aplicaciones el programador de lenguaje máquina puede llamar a las rutinas kernel en vez de diseñar las suyas propias. No obstante, en algunas ocasiones resulta útil programar el PIO directamente, como para emplear los temporizadores PIO o utilizar la puerta para el usuario

Par PIO

Programando el PIO

El chip PIO es programable, es decir, posee registros internos a los que el procesador principal puede acceder a través de su sistema de direccionamiento normal. Colocando ciertos valores en estos registros se puede hacer que el PIO se comporte de ciertas maneras. Por ejemplo, se pueden programar independientemente cada una de las ocho líneas de datos que componen cada puerta E/S, de modo que actúen como líneas de entrada o salida mediante un registro de dirección de datos. En consecuencia, se puede hacer que una misma puerta sea para entrada y salida simultáneamente utilizando diferentes bits. De este modo, diseñando un software manejador apropiado, un PIO se puede aplicar a la mayoría de los problemas de conexión en interface. Además, debido a que las operaciones de E/S se han de realizar a intervalos rígidamente definidos, los PIO incorporan temporizadores de cuenta atrás de 16 bits que se pueden cargar con un valor y reducir con cada impulso del reloj del sistema, generando una señal de interrupción cuando se produce un desbordamiento negativo (el contador intenta reducir más atrás de cero) del temporizador. En consecuencia, a través de los temporizadores PIO se pueden controlar las comunicaciones de datos síncronas, en virtud de las cuales se transmiten o reciben datos a intervalos regulares

Tres grandes

Analizaremos tres paquetes que han tenido gran aceptación entre los usuarios personales

El tratamiento de textos es, indudablemente, una de las aplicaciones para microordenadores personales más conocidas. Los programas van desde sencillos editores de textos de precio muy reducido hasta sofisticados y costosos paquetes de tratamiento de textos basados en ROM o en disco. Sin embargo, la bondad de cualquier programa está en relación directa con la del hardware para el cual ha sido diseñado, y los paquetes escritos para la mayoría de los micros personales carecen de muchas de las facilidades más complejas que ofrecen los programas para ordenadores de gestión.

La restricción fundamental de un procesador de textos para un ordenador de ocho bits viene dada por las limitaciones de memoria. Toda facilidad adicional incluida en el software a menudo ocupa un espacio que, en otras circunstancias, estaría disponible para almacenar texto. Esto ha demostrado ser un obstáculo importantísimo en la implementación del *WordStar* para el Amstrad CPC 464 y 664. Si bien estas máquinas pueden usar el sistema operativo CP/M, la memoria disponible es insuficiente para implementar adecuadamente el programa *WordStar* completo. No obstante, se ha desarrollado una versión a pequeña escala del programa, denominada *Pocket WordStar* (WordStar de bolsillo).

Otro problema con el que tropiezan los programadores de procesadores de textos para micros personales es el método de almacenamiento de apoyo. La mayoría de los micros personales continúan basándose en cassettes para el almacenamiento de apoyo; por lo tanto, tanto el programa como el texto se han de retener enteramente en la memoria del ordenador, imponiendo demandas aún mayores en la memoria disponible.

"Easy script"

Para el Commodore 64 se han escrito muchos paquetes para tratamiento de textos, pero posiblemente el más difundido de ellos sea el *Easy script* (debido fundamentalmente a que Commodore lo suministra empaquetado con la unidad de disco).

El principal problema para quien desee escribir un paquete de tratamiento de textos para el Commodore 64 es la visualización en pantalla, que sólo puede producir una resolución de caracteres máxima de 40 por 25. En consecuencia, el programador se ve obligado a sacar el máximo provecho de las 40 columnas, lo que significa que el *Easy script* no cuenta con el lujo que suponen las pantallas de ayuda, que ocuparían demasiado espacio en la pantalla. Además, las limitaciones de pantalla impiden que el *Easy script* posea alguna facilidad para desplazamiento de palabras, aunque es posible contemplar el texto en formato definitivo antes de enviarlo a la impresora.

Liz Heaney

Para superar este problema, los programadores del *Easy script* han optado por utilizar la pantalla como una "ventana" del documento. Los márgenes se pueden fijar en 240 columnas a lo ancho, y cuando una línea llega al límite de las 40 columnas la pantalla se desplaza lateralmente hasta llegar a la limitación del margen. En este punto, el retorno a la siguiente línea resultará un tanto abrupto. Si bien este método no es el ideal, el usuario suele acostumbrarse fácilmente al mismo o, de lo contrario, puede escribir en 40 columnas y después reformatear el texto antes de enviarlo a la impresora.

A las diversas instrucciones de formateo y edición de texto para impresión que son esenciales para un programa de tratamiento de textos se accede a través de las teclas de función del Commodore 64. Pulsando una de ellas usted entrará en una modalidad determinada y, si a continuación efectúa una pulsación de tecla, se seleccionará la opción adecuada. Por tanto, a las instrucciones de edición se accede pulsando F1, lo que se indica mediante la palabra Edit, que aparece intermitentemente en la parte superior de la pantalla. El trazado del texto y la página se seleccionan pulsando F3, que produce un asterisco invertido en la posición del cursor. A continuación se pueden determinar los códigos para establecer los márgenes, ajustar el texto, etc. *Easy script* hace un uso intensivo de estas y otras instrucciones incorporadas en el texto, las cuales sólo actúan cuando se envía el texto a la impresora. Esto significa que muchas de las funciones no se verán en la pantalla, reduciendo la capacidad de ver el aspecto que tendrá el documento final impreso.

Easy script contiene casi todas las facilidades que cabría esperar de un paquete de tratamiento de textos, tales como manipulación de bloques enteros, lo que incluye la copia, transferencia y archivado separado de los bloques, si bien hay que destacar que algunas de las facilidades funcionan mejor que otras. La función Buscar/Reemplazar, por ejemplo, es notablemente lenta, en especial cuando se utilizan archivos unidos que se han cargado por separado y que luego se han buscado.



"Tasword II"

Un programa para tratamiento de textos que se utiliza ampliamente en la gama Amstrad y en el Spectrum+ es el *Tasword II*. Al cargarlo, este programa guarda un sorprendente parecido con el *WordStar*. En la parte superior de la pantalla hay un menú de Ayuda que contiene una lista de las instrucciones (y sus caracteres de control), dividida en secciones lógicas. Debajo de este menú está la zona de 16 por 80 caracteres reservada para escribir el texto.

El *Tasword II* es un potente paquete para tratamiento de textos. El programa contiene una gran cantidad de instrucciones que permiten que el usuario formatee el texto en la pantalla. Éste se puede desplazar hacia la derecha, la izquierda o centrar, y se pueden establecer los márgenes hasta un máximo de 128 columnas. Si se vuelven a establecer los márgenes, se puede reajustar el texto de modo que quepa en el nuevo formato. El *Tasword* tiene implementado el desplazamiento de palabras, reajustando la línea de forma automática si se lleva una palabra hasta la línea siguiente. El programa es inusual en cuanto que los espacios adicionales sólo se incluyen en la segunda mitad de la línea.

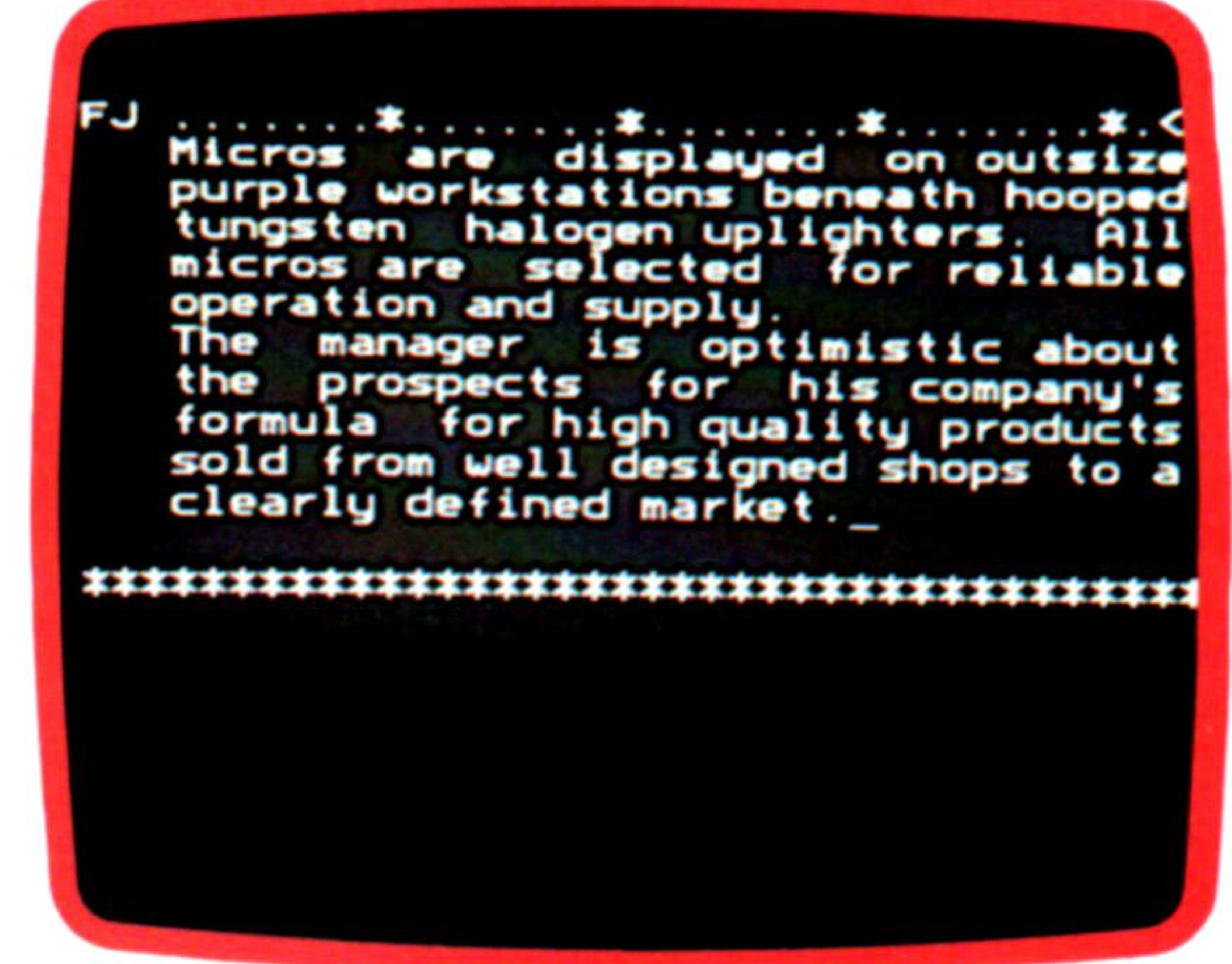
El programa también facilita instrucciones de bloques, si bien la lista de ellas no es muy amplia. Una vez definidos, los bloques se pueden desplazar, copiar o borrar, pero las instrucciones adolecen de un problema que se plantea en varias características del *Tasword II*: el programa delimita los bloques en conjuntos de líneas. De este modo, cuando el usuario define un bloque que comience a medio camino de una línea, el programa suele desplazar la línea entera, incluyendo las palabras que supuestamente habrían de quedar fuera del área del bloque.

Cuando se utiliza la modalidad insertar se plantea un problema similar. Muchos procesadores de textos insertan automáticamente el texto en la posición del cursor y desplazan el resto del texto hacia la derecha. Lamentablemente, el *Tasword* permite ya sea insertar un único carácter, o bien insertar una línea entera. Además, esta línea extra suele aparecer debajo de la actual, ignorando nuevamente el hecho de que el cursor pueda hallarse en mitad de una línea y no al final de la misma.

Donde el *Tasword* se destaca verdaderamente es en la gran cantidad de facilidades para control de impresora que ofrece. En primer lugar, los usuarios pueden adaptar a su medida, o "instalar", su propia versión de Tasword.

El *Tasword* posee una amplia gama de caracteres de control para la impresora basados en los códigos Epson estándares. A ellos se accede pulsando CTRL y la barra Space al mismo tiempo, y a continuación la letra adecuada, con las mayúsculas activando la opción y las minúsculas desactivándola. Hay 20 códigos disponibles en el programa, que van desde espaciado entre líneas hasta diversas formas de impresión resaltada, como subrayado, negritas y cursiva.

Los caracteres de control de la impresora Tasword proveen caracteres fuente adicionales, como Lectura Light y Compacta, pero los mismos no están disponibles en el programa propiamente dicho. Para poder acceder a ellos, Tasman, el fabricante, ha introducido un segundo programa que se puede ejecutar en el *Tasword* y se denomina *Tasprint*, que se puede configurar para una amplia gama de impresoras matriciales. Una vez configurado, el *Tasprint* se puede cargar antes que el *Tasword* y se asigna a sí mismo una zona de la memoria sobre la cual no escribirá el *Tasword*, tras lo cual el usuario puede acceder a las cinco fuentes adicionales. Como es natural, al hacer uso de estas fuentes adicionales se reduce la zona de textos; *Tasprint* ocupa alrededor de 5 de los 13 Kbytes que estarían disponibles en caso contrario.



"View"

El problema a superar en el caso del BBC Micro es ligeramente distinto al del Commodore 64. Su visualización en pantalla utiliza una gran cantidad de memoria que, en el Modelo B estándar, es escasa. Al igual que numerosas otras empresas, AcornSoft ha optado por suministrar su paquete para tratamiento de textos, *View*, en ROM para instalar en uno de los conectores laterales del BBC Micro. Éste no ocupa nada del espacio de memoria disponible y, al mismo tiempo, evita el problema con el que se enfrentan algunos de los sistemas grandes de tratamiento de textos, que han de leer continuamente programas extras desde disco.

Ésta es una consideración importante. Si usted decidiera contar con una visualización de 80 columnas, lo mejor que se podría esperar son algo menos de 10 000 caracteres en memoria. Si en ella se incluyera un programa de ocho Kbytes, el área de textos se reduciría considerablemente, pero con el *View* no es necesario utilizar la modalidad de 80 columnas. La pantalla se puede emplear como una ventana sobre el documento, como con el *Easy script* y la mayoría de los otros paquetes.

El *View* es sin duda uno de los mejores programas para tratamiento de textos para micros. En lugar de tener la pantalla de ayuda, se proporciona al usuario una plantilla que se coloca sobre las teclas de función, dado que a casi todas las instrucciones de tratamiento de textos se accede a través de estas teclas, ya sea individualmente o conjuntamente con las teclas Shift o Control.

Entre las facilidades adicionales que incluye el *View* está la facilidad que permite al usuario definir sus propias *macros*. Éstas son programas cortos (creados utilizando instrucciones del *View* y texto) que se pueden emplear para crear instrucciones adicionales mediante un código de dos letras definido por el usuario.

Asimismo el programa incluye instrucciones que llevan a cabo funciones tales como cuenta de palabras, formateo, proceso continuo (que permite editar un documento desde disco o cinta y después pasarlo directamente a otro archivo) y un juego completo de instrucciones Buscar/Reemplazar. Éstas no sólo realizan las funciones habituales, sino que permiten caracteres.



WordStar de bolsillo

Una muestra de la seriedad con la que en la actualidad se está considerando a la gama de ordenadores Amstrad está en el hecho de que varias empresas de software que previamente sólo fabricaban programas para aplicaciones de gestión, ahora están "adaptando" sus programas CP/M para ejecutarlos en máquinas Amstrad. Una de las jugadas maestras de la empresa de Alan Sugar es que MicroPro ha desarrollado una versión del *WordStar*, el líder de los paquetes profesionales para tratamiento de textos, para ejecutar en los ordenadores CPC 464 y 664. Aunque el *Pocket WordStar* opera con menos memoria que el paquete original, esto no significa que haya perdido muchas de las facilidades del programa madre. Por el contrario, en el *Pocket WordStar* se han implementado también casi todas las facilidades que proporciona el paquete *WordStar* completo. Las únicas que le faltan a la versión hecha a medida para Amstrad son aquellas que requieren más memoria de la que hay disponible en el ordenador; p.ej., la facilidad que permite al usuario imprimir (Print) un archivo (File) y editar (Edit), al mismo tiempo. MicroPro ha logrado conseguir esto al retener en disco varios submenús, que se cargan cuando se necesitan. Esto, por supuesto, significa que el *Pocket WordStar* es algo más lento que su predecesor, pero ésta es una pequeña contrapartida a cambio de un paquete de tratamiento de textos excelente para una máquina de costo tan económico

EASY SCRIPT

Desplazamiento de palabras

Easy script sólo lo soporta cuando el documento se está enviando a la impresora.

Movimiento de bloques



Admite transferencia, copiado y archivado separado.

Ayuda en pantalla

No tiene.

Pantalla de 80 columnas

Easy script posee una pantalla de 40 columnas.

Contador de palabras

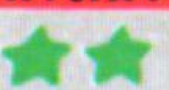
Easy script no posee facilidad para contar las palabras.

Buscar/Reemplazar



Se pueden buscar series de hasta 32 caracteres, si bien la facilidad es lentísima.

WYSIWYG



Aunque el *Easy script* permite examinar el trazado de la página, la baja resolución de pantalla limita su utilidad.

Facilidad para correspondencia



Incorpora una facilidad Mail Merge simple, pero no permite imprimir etiquetas de direcciones.

Verificador de ortografía



Los mismos fabricantes ofrecen un paquete hermano: *Easy spell*.

Fuentes disponibles



Easy script soporta impresión agrandada, condensada, invertida y en negritas.

Unión de archivos

El programa soporta instrucciones para unir archivos, ya sea desde cassette o disco, para editar o imprimir.

TASWORD II

Desplazamiento de palabras



Completo y con ajuste automático.

Movimiento de bloques



Permite funciones para transferir, copiar y suprimir, pero no archivado separado.

Ayuda en pantalla



Se proporciona mediante una ventana en la parte superior de la pantalla, que se puede desplazar para visualizar todas las instrucciones disponibles.

Pantalla de 80 columnas

El programa visualiza una pantalla completa de 80 columnas, con un valor máximo para el margen derecho de 128.

Contador de palabras

Disponible constantemente.

Buscar/Reemplazar



Tasword II sólo permite la búsqueda de una única palabra.

WYSIWYG



Hay varias opciones de formato de impresión que sólo la impresora reconoce como caracteres de control; no se pueden visualizar en la pantalla.

Facilidad correspondencia

No tiene.

Verificador de ortografía

No tiene.

Fuentes disponibles



Tasword II implementa varias fuentes diferentes; el *Tasprint* proporciona estilos adicionales.

Unión de archivos



El programa sólo permite añadir un archivo en la cola de otro.

VIEW

Desplazamiento de palabras

Soporta desplazamiento de palabras completo, aunque no hay instrucciones para desactivar esta función.

Movimiento de bloques



Las operaciones de bloques que permite incluyen supresión, transferencia y copia.

Ayuda en pantalla



No ofrece ayuda en pantalla, pero proporciona una plantilla para cubrir las teclas de función.

Pantalla de 80 columnas



View soporta una pantalla de 76 columnas. Sin embargo, ésta reduce drásticamente la cantidad de memoria que queda para el texto.

Contador de palabras



Si bien posee una facilidad para contar las palabras (que en realidad cuenta los espacios), no es una visualización constante.

Buscar/Reemplazar



View sólo puede hallar y reemplazar individuales, si bien soporta caracteres *wildcard*

WYSIWYG



El programa permite la visualización del documento tal como aparecerá en la página, exceptuando las facilidades resaltadas (subrayando la impresión en negrita).

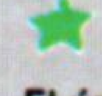
Facilidad correspondencia

No tiene.

Verificador de ortografía

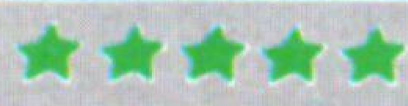
No tiene, pero AcornSoft ha prometido que en el futuro proporcionará uno.

Fuentes disponibles



El único énfasis adicional disponible son las negritas.

Unión de archivos



Las instrucciones Macro permiten una amplia gama de facilidades para unir archivos.

Mudar de sitio

Nos quedan por examinar cuatro modos de direccionamiento y mostrar sus diferencias con algunos ejemplos en código máquina

Para empezar hagamos un repaso de los modos que ya conocemos:

- *Direccionamiento absoluto*: En este modo el operando fuente o destino es una dirección de memoria.
- *Direccionamiento con registro*: La fuente o destino es un registro.
- *Direccionamiento indirecto con registro (y puntero)*: Apuntamos al objeto fuente o destino.
- *Direccionamiento indirecto con registro (y postincremento o predecremento)*: Recorremos una lista de datos de arriba abajo o viceversa.

Consideremos ahora el modo *inmediato*, en el que almacenamos una constante próxima a la instrucción.

- *Direccionamiento inmediato*: En este modo el operando fuente es una constante. Por ejemplo, MOVE.W #$43,D0 ordena que la constante (especificada mediante el símbolo #) $43 ($ significa hexadecimal) es llevada a D0. Se dice que es un direccionamiento inmediato porque la palabra constante se almacena en la propia instrucción MOVE. Es obvio que para operandos bytes la extensión toma un byte, y para palabras largas toma cuatro bytes.

Este modo inmediato es muy útil para establecer una constante, por ejemplo para un bucle con cuatro reiteraciones. Pero es de notar que pueden darse ocasiones en que sea mejor definir una constante en una posición determinada y emplear el direccionamiento absoluto hacia esta posición cada vez que se necesite. Por ejemplo:

```
MOVE    COUNT,D0
MOVE    COUNT,D5
COUNT   DC.W 4
```

Esto puede que se prefiera al direccionamiento en modo inmediato cada vez que se prevea que el valor COUNT ha de cambiar. Con el método absoluto cambiamos el contenido de COUNT; de otra manera tendríamos que investigar todo el código para todas las referencias en modo inmediato, empleando, por ejemplo, 4, lo cual, además de ser aburrido, puede comportar errores.

Es posible un modo inmediato *rápido* con ciertas instrucciones, donde la constante está contenida dentro de las instrucciones en una palabra. Tanto uno como el otro de estos dos ejemplos:

```
ADDQ    #3,D0
SUBQ    #1,D3
```

se codificarían en una palabra (y, por tanto, se ejecutarán más rápidamente ya que tienen la constante junto con la instrucción). Obsérvese, sin embargo, que las constantes con ADDQ y SUBQ sólo pueden estar en el intervalo #1 a #8.

La instrucción MOVE tiene también una versión rápida en la que el ámbito de las constantes se mueve entre −128 y +127. Así, MOVEQ #98,D4 pone en D4 el decimal 98.

Para reforzar su memoria, he aquí un ejemplo donde se emplean todos los modos de direccionamiento analizados hasta aquí:

1		LEA	OUTPUT,A1	Absoluto y registro
2		MOVE	A1,A2	Registro
3		MOVE	−(A1),D3	Predec. y registro
4		ADDQ	#3,D3	Inmediato rápido y registro
5		MOVE	D3,(A2)+	Registro y postinc.
6		ADD	#6,D3	Inmediato y registro
7		MOVE	D3,(A2)	Registro e indirecto
8				
9	INPUT	DC.W	#6	Constante 6
10	OUTPUT	DS.W	2	Espacio para dos palabras

La posición INPUT contiene una constante de 6 y la dirección OUTPUT tiene un espacio para dos palabras.

Interesa tener en cuenta lo que está contenido en OUTPUT y OUTPUT+2 tras la ejecución de este fragmento en código máquina. Las líneas 1 y 2 hacen que A1 y 2 apunten a OUTPUT. La línea 3 hace que A1 apunte a INPUT antes de direccionar su operando, por lo que en D3 se cargará 6. La línea 4 añadirá 3 a D3, o sea el contenido de D3 es ahora 9.

En la línea 5, D3 se cargará en OUTPUT, ya que A2 apunta a OUTPUT. Una vez hecho esto, A2 se incrementa en dos para apuntar a la segunda palabra OUTPUT. La línea 6 suma 6 al contenido de D3 (contenido total, 15), que será cargado en la segunda palabra OUTPUT.

- *Direccionamiento indirecto (con desplazamiento e índice)*: Debemos considerar primero el significado que aquí se da a los términos *desplazamiento* (*displacement*) e *índice* (*index*). Con el 68000 por *desplazamiento* se entiende una "distancia" u *offset* fijo desde un punto de referencia básico; por su lado, *índice* significa un desplazamiento variable a través de un registro.

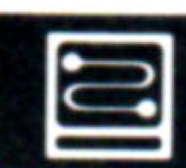
La diferencia entre estos dos términos queda ilustrada en el dibujo (página contigua). En él se muestran:

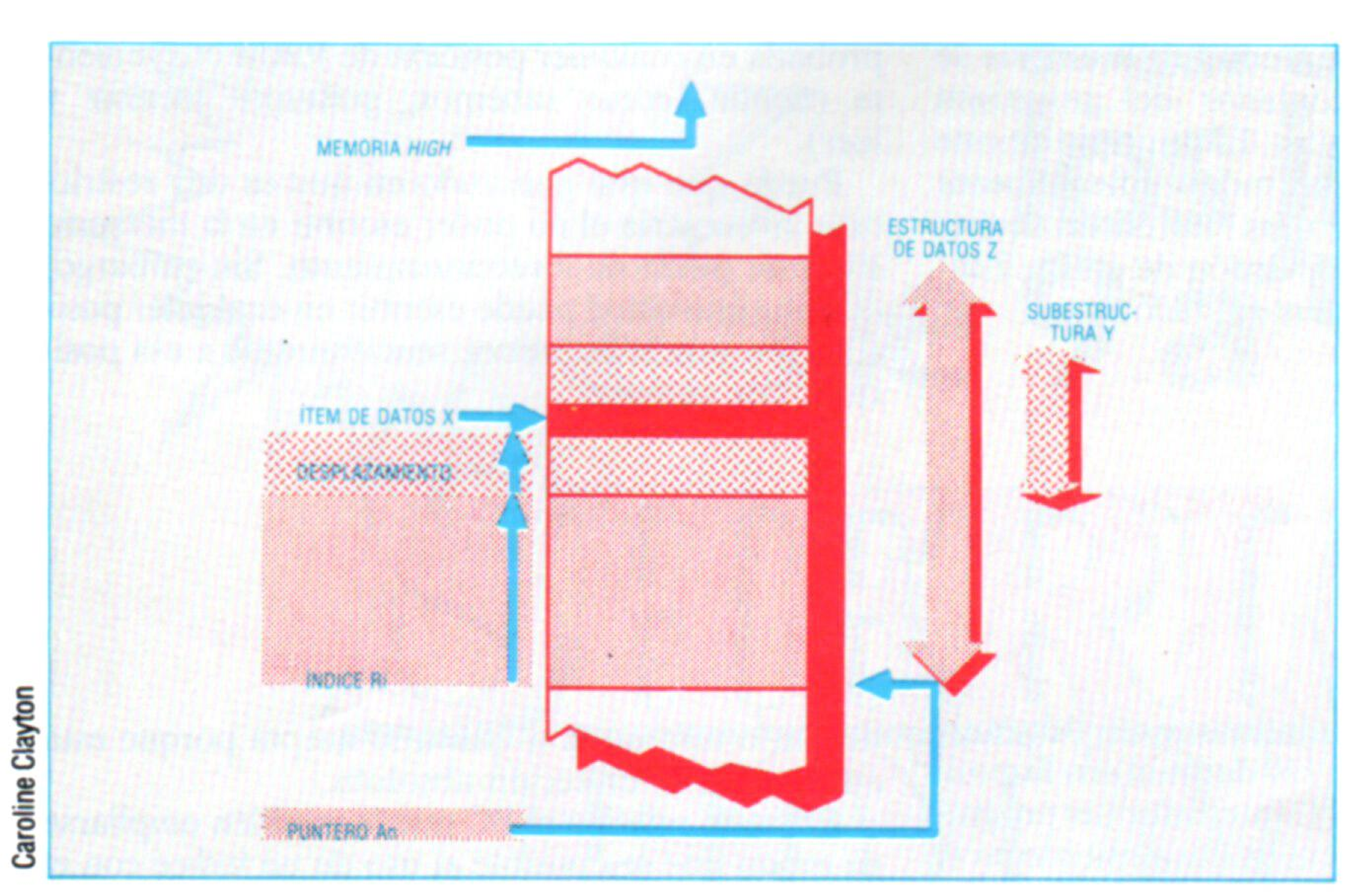
1) un componente de datos estructurados llamado Z, al que apunta el registro de direcciones An;
2) una subestructura interna llamada Y, indexada mediante un registro índice llamado Ri (registro de direcciones o de datos);
3) y un elemento de Y llamado X, al que se puede acceder con un desplazamiento fijo.

Ejemplos de estas estructuras de datos:



Saberse el camino
La *Guía del usuario* del 68000 contiene breves detalles de las instrucciones y modos del 68000, pero donde resulta más útil es en las sugerencias sobre cómo sacar provecho de las facilidades de éste en numerosas aplicaciones de ejemplo. El libro ha sido publicado por la editorial británica Sigma Press





Caroline Clayton

1) Z: tabla de registros, o tabla de dimensiones;
2) Y: el registro mismo, o bien una fila de una tabla;
3) X: elemento de un registro, o bien el entero de una tabla.

Se trata de tres componentes de datos estructurados que son conocidos en los modernos lenguajes de alto nivel tipo PASCAL y ADA, y, por tanto, es importante poder referirnos fácilmente a tales componentes. Naturalmente, no hay razón alguna por la que el programa ensamblador no pueda estructurar los datos de modo similar: ¡sin duda disponemos de medios para hacer esto con el 68000!

Volvamos una vez más a nuestro ejemplo de datos estructurados. Habrá notado que la dirección de X está formada por la suma An+Ri+desplazamiento, y que podemos alterar An y Ri conforme se ejecuta el programa. Podemos mirar ahora algunos ejemplos de direccionamiento, y comenzaremos con uno que lo ilustra cabalmente en su forma más sencilla (direccionamiento indirecto con desplazamiento):

```
MOVE.W DISP (A0),D1
```

donde DISP habrá sido definido previamente como nombre simbólico, supongamos, de 6. Esta instrucción emplea el direccionamiento indirecto con un desplazamiento de 6 como modo de direccionamiento fuente. Si, por ejemplo, A0 estuviera apuntando a la dirección $1000 (1000 hexa), entonces el contenido de la dirección $1006 se cargaría en D1. Nótese que al aceptarse una extensión de palabra entera con la instrucción para el desplazamiento, podemos tener un entero con signo de 16 bits completos para desplazamiento (valores entre +32767 y −32768).

Veamos ahora la modalidad de direccionamiento más poderosa que tiene el programador del 68000 (el desplazamiento), considerada en este ejemplo:

```
MOVE.W DISP (A0,D0.W),D1
```

En este caso, la dirección fuente está formada añadiendo juntamente el registro base, A0, con el registro índice, D0, y el desplazamiento fijo, DISP. Es de notar, sin embargo, que en este caso DISP sólo puede ser un entero con signo de ocho bits, aunque el registro índice puede ser una palabra larga de 32 bits completos.

Obsérvese que cuando usted compara estos dos modos de direccionamiento (el indirecto y el desplazamiento, con o sin índice), el desplazamiento siempre es fijo en tiempo de ensamblaje. Sin embargo, si usted necesita alterar dinámicamente un desplazamiento, entonces podrá optar siempre por usar:

```
MOVE.W 0(A0,D1),D2
```

donde D1 se convierte ahora en el desplazamiento de hecho (que podemos alterar en tiempo de ejecución del programa), y el desplazamiento fijo es cero.

Finalmente es oportuno señalar que el conjunto de instrucciones del 68000 ofrece estos dos potentes modos de direccionamiento para la mayoría de las instrucciones más comunes. Veamos un ejemplo de direccionamiento que emplea desplazamientos e índices, con bytes como atributos de datos.

```
1          LEA      LIST,A1
2          MOVEQ    #4,D1             D1:=4
3          MOVE.B   2(A1),D6          D6:=3
4          ADD.B    0(A1,D1),D6       D6:=8
5          MOVE.B   D6,4(A1,D1)       LIST+8:=8
6
7    LIST  DC.B     1,2,3,4,5,6,7,8,0
```

A1 es puesto de modo que apunte a LIST en la línea 1 y seguidamente D1 es puesto a 4. En la línea 3, al puntero que hay en A1 se añade un desplazamiento de 2, de forma que el tercer elemento de LIST es cargado en D6. Después, con un índice de 4 en D1, se añade a D6 el contenido de elemento 5, siendo copiado en LIST+8.

Obsérvese que en la línea 4 hemos empleado el desplazamiento de 0: esto permite emplear D1 como registro índice, que podemos alterar cuando se ejecuta el programa, si fuera el caso.

• *Direccionamiento relativo al PC*: Antes de pasar a analizar en detalle los modos de direccionamiento relativo al PC debemos echar un vistazo a algunas directivas del ensamblador. Las directivas son instrucciones dadas al ensamblador que no producen directamente código ejecutable, pero que influyen en factores tales como el formato del listado del programa fuente o la definición de símbolos. Una de estas directivas se refiere específicamente a la dirección de inicio del programa y el tipo de código producido. Es la directiva ORG, o sea origen.

Normalmente se especificará la dirección de inicio, por ejemplo así:

```
ORG    $1000
```

Con ello establecemos la dirección de inicio en $1000 (en realidad es la dirección de carga para el cargador binario), y todo el código que sigue es cargado en direcciones secuencialmente crecientes. Si aparece una nueva directiva ORG, establecerá entonces una nueva dirección de carga en ese punto. Una variante de esto sería:

```
ORG.L    $2000
```

donde todas las referencias posteriores se tomarán como palabras largas completas, y por ende cualquiera de estas referencias ocupará dos palabras enteras.

La directiva RORG define una dirección de carga



Enseñanza general
Este libro sobre programación en lenguaje assembly del 68000 es una guía completa sobre el tema publicado por la editorial británica Osborne/McGraw-Hill. Los programadores experimentados encontrarán algo superfluas las secciones sobre teoría de la programación y el estilo del libro es enfáticamente magistral. Pero si por un lado le falta informalidad por otro incluye toda la información que usted pueda necesitar. Un buen detalle es el empleo de encabezamientos de párrafo en mayúsculas, lo que permite buscar rápidamente un tema concreto a través de las secciones

desde la cual todas las referencias de memoria se consideran relativas al contador del programa (PC), dejando de ser absolutas. El principio de este tipo de código se basa en que, independientemente de donde se cargue el código, las referencias de memoria corresponden a esa dirección de carga. Pongamos por ejemplo el siguiente listado:

```
                          RORG   $2000
2000   D47A    START      ADD    CONST1,D2
2002   000A
2004   3202               MOVE   D2,D1
2006   6000               BRA    START
2008   FFFB
200A   4E40               TRAP   #0
200C   0026    CONST1     DC.W   38
```

Aquí, CONST1 está referenciado con un desplazamiento de 16 bits de A hexa (10 decimal) en la posición 2002, que se añade al PC (contador del programa). Para cuando se cargue el desplazamiento en el ejemplo, el PC será 2002 de modo que la dirección de datos fuente será 200C. Obsérvese que no hay instrucciones específicas o diferencias en la sintaxis de direccionamiento para obtener el código relativo al PC; todo lo que se requiere es la directiva RORG del ensamblador.

Hay unos cuantos puntos más sobre este código dignos de nota. Primero, que la instrucción BRA (*branch always*: bifurcar siempre) tiene un desplazamiento asociado con ella en la dirección 2008, el cual, una vez añadido al PC, dará la dirección de operando START. Esto significa que la instrucción BRA dará siempre el código relativo al PC. El segundo punto a notar es la instrucción TRAP. Se usa en este contexto como una instrucción de parada, con el añadido de que la entrada se hace a una pantalla que imprime los registros y permite al usuario el examen de la memoria. Esta instrucción, si su pantalla se lo permite, puede resultar muy útil.

Veamos otro ejemplo, donde el direccionamiento de modo relativo al PC da un desplazamiento negativo:

```
                          RORG   $2000
2000   0026    CONST1     DC.W   38
2002   DA7A    START      ADD    CONST1,D2
2004   FFFC
2006   3202               MOVE   D2,D1
2008   6000               BRA    START
200A   FFF8
200C   4E40               TRAP   #0
```

Aquí, la referencia a CONST1 es un desplazamiento negativo en la dirección #2004. (Naturalmente, usted conocerá que FFFC es negativo porque el bit de signo estará activado. La magnitud, o tamaño, de los números pueden encontrarse invirtiéndolos y añadiendo uno; por tanto, FFFC es el decimal −6).

Una limitación a este direccionamiento es que solamente se permite como operando fuente. Por ejemplo:

```
         RORG   $2000
TOTAL    DC.W   0
START    ADD    D2,TOTAL
```

generará un error de ensamblador. Esto significa que este tipo de direccionamiento es muy conveniente para el código que ha de ponerse en una ROM (dado que no podemos escribir en las ROM) en una dirección fija, pero todavía puede ser comprobada en cualquier posición de RAM conveniente (donde, como sabemos, podemos escribir y leer).

Puede que esté pensando en que es una restricción innecesaria el no poder escribir en la memoria con este modo de direccionamiento. Sin embargo, observe que usted puede escribir en cualquier posición absoluta refiriéndose sencillamente a esa posición. Por ejemplo:

```
         ORG    $1000
TOTAL    DS.W   0
         RORG   $2000
CONST1   DC.W   38
START    ADD    CONST1,D2
         MOVE   D2,TOTAL
```

donde la referencia a TOTAL se acepta porque está en un área de dirección absoluta.

El modo relativo al PC puede también empliarse de modo que sea posible el uso de un índice con el desplazamiento.

Por ejemplo:

```
         RORG   $3000
INDEX    DC.W   10
START    MOVEA  INDEX,A5
         ADD    6(A5),D2
```


En este caso tenemos que usar la instrucción MOVEA, que carga A5 con el contenido de la posición de memoria INDEX. No podríamos usar LEA porque tomaría la dirección de INDEX (de ningún modo permisible en el modo relativo al PC). Es claro que una instrucción MOVE directa sería ilegal, como ya hemos visto (un registro de dirección no es legal como operando destino).

Volvamos ahora al ejemplo donde el operando fuente de la instrucción ADD será el valor de PC después de la instrucción ADD más el registro índice (aquí, A5) con un desplazamiento de 6. En este ejemplo, la dirección fuente será 16 bytes más allá de la dirección que contiene la palabra de extensión 6.

Una limitación de este modo es, sin embargo, el que el desplazamiento esté formado por ocho bits dentro del opcode, limitando nuestro desplazamiento a un número de bytes desde +127 a −128.

La importancia de este modo relativo al PC es que el código puede ejecutarse en cualquier sitio de la memoria, como hemos visto en el caso de escribir código para la ROM. Obsérvese, sin embargo, que esta forma de direccionamiento es extremadamente útil, por lo general, para escribir código independiente de la posición en memoria. Esta forma de código puede ser necesaria para extensos programas de muchos módulos donde la posición de un módulo en la memoria no es fijada hasta que el módulo se carga en la memoria. Naturalmente, para largos sistemas de multiprogramación, puede que necesitemos una unidad de gestión de la memoria con facilidades adicionales (que proporciona Motorola), pero para cualquier esquema de memoria más sencillo el modo relativo al PC es muy importante.

- *Direccionamiento implícito*: Este modo de direccionamiento no debería ser dificultoso, dado que el opcode especifica los registros que hay que usar. Por ejemplo, la instrucción RTS afectará al PC y al puntero de la pila; BRA afecta al PC.


NUEVO
Para los jóvenes de 9 a 90 años
SHERLOCK HOLMES
Una nueva y divertida serie de tve realizada en COMICS forum
PÍDELA EN TU QUIOSCO
COPYRIGHT © 1986 RAI/PAGOT/TMS
A-048 B


GRAN NOVEDAD EDITORIAL
PARA TI, PROFESIONAL EN ACTIVO.....
PARA TI, FUTURA SECRETARIA.....
enciclopedia de la
SECRETARIA
de
PLANETA - AGOSTINI
PIDELA EN TU QUIOSCO
o suscríbete ahora llamando al (91) 415 97 12
... para estar al día
... para ser más eficaz
... para actualizar tus estudios
... para encontrar mejor empleo